Modeling Cardiovascular Hemodynamics Using the Lattice Boltzmann Method on Massively Parallel Supercomputers

Modeling Cardiovascular Hemodynamics Using the Lattice Boltzmann Method on Massively Parallel Supercomputers

A dissertation presented

by

Amanda Elizabeth Randles

to

School of Engineering and Applied Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Applied Physics

Harvard University

Cambridge, Massachusetts

May 2013

Thesis advisor

Efthimios Kaxiras

Author

Amanda Elizabeth Randles

Modeling Cardiovascular Hemodynamics Using the Lattice Boltzmann Method on Massively Parallel Supercomputers

Abstract

Accurate and reliable modeling of cardiovascular hemodynamics has the potential to improve understanding of the localization and progression of heart diseases, which are currently the most common cause of death in Western countries. However, building a detailed, realistic model of human blood flow is a formidable mathematical and computational challenge. The simulation must combine the motion of the fluid, the intricate geometry of the blood vessels, continual changes in flow and pressure driven by the heartbeat, and the behavior of suspended bodies such as red blood cells. Such simulations can provide insight into factors like endothelial shear stress that act as triggers for the complex biomechanical events that can lead to atherosclerotic pathologies. Currently, it is not possible to measure endothelial shear stress *in vivo*, making these simulations a crucial component to understanding and potentially predicting the progression of cardiovascular disease. In this thesis, an approach for efficiently modeling the fluid movement coupled to the cell dynamics in real-patient geometries while accounting for the additional force from the expansion and contraction of the

heart will be presented and examined.

First, a novel method to couple a mesoscopic lattice Boltzmann fluid model to the microscopic molecular dynamics model of cell movement is elucidated. A treatment of red blood cells as extended structures, a method to handle highly irregular geometries through topology driven graph partitioning, and an efficient molecular dynamics load balancing scheme are introduced. These result in a large-scale simulation of the cardiovascular system, with a realistic description of the complex human arterial geometry, from centimeters down to the spatial resolution of red-blood cells. The computational methods developed to enable scaling of the application to 294,912 processors are discussed, thus empowering the simulation of a full heartbeat.

Second, further extensions to enable the modeling of fluids in vessels with smaller diameters and a method for introducing the deformational forces exerted on the arterial flows from the movement of the heart by borrowing concepts from cosmodynamics are presented. These additional forces have a great impact on the endothelial shear stress. Third, the fluid model is extended to not only recover Navier-Stokes hydrodynamics, but also a wider range of Knudsen numbers, which is especially important in micro- and nano-scale flows. The tradeoffs of many optimizations methods such as the use of deep halo level ghost cells that, alongside hybrid programming models, reduce the impact of such higher-order models and enable efficient modeling of extreme regimes of computational fluid dynamics are discussed. Fourth, the extension of these models to other research questions like clogging in microfluidic devices and

determining the severity of co-arctation of the aorta is presented. Through this work, a validation of these methods by taking real patient data and the measured pressure value before the narrowing of the aorta and predicting the pressure drop across the co-arctation is shown. Comparison with the measured pressure drop *in vivo* highlights the accuracy and potential impact of such patient specific simulations.

Finally, a method to enable the simulation of longer trajectories in time by discretizing both spatially and temporally is presented. In this method, a serial coarse iterator is used to initialize data at discrete time steps for a fine model that runs in parallel. This coarse solver is based on a larger time step and typically a coarser discretization in space. Iterative refinement enables the compute-intensive fine iterator to be modeled with temporal parallelization. The algorithm consists of a series of prediction-corrector iterations completing when the results have converged within a certain tolerance. Combined, these developments allow large fluid models to be simulated for longer time durations than previously possible.

Contents

Citations to Previously Published Work

Large portions of Chapter 4, appeared in the following paper:

> "Multiscale simulation of cardiovascular flows on the IBM Blue Gene/P: full heart-circulation system at red-blood cell resolution", A. Peters, S. Melchionna, E. Kaxiras, J. Latt, J. Sircar, M. Bernaschi, M. Bisson, and S. Succi, Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), New Orleans, LA.

Large portions of Chapter 5, will appear in the following paper:

> "Performance analysis of the Lattice Boltzmann model beyond Navier-Stokes", A. Peters Randles, V. Kale, J.R. Hammond, W. Gropp, and E. Kaxiras, Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium, IPDPS 13, 2013.

Large portions of Chapter 7 are from the following two papers:

> "Parallel in Time Approximation of the Lattice Boltzmann Method for Laminar Flows", A. Peters Randles and E. Kaxiras, submitted to Journal of Computational Physics.

and

> "Reduction in Time to Solution for Modeling Patient Specic Cardiovascular Hemodynamics through Space-Time Parallelization", A. Peters Randles and E. Kaxiras, submitted to Proceedings of the 2013 ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), Denver, CO.

Large portions of Chapter 6, appear in the following paper:

> "A Lattice Boltzmann Simulation of Hemodynamics in a Patient-Specific Aortic Coarctation Model", A. Peters Randles, M. Baecher, H. Pfister, and E. Kaxiras, STACOM 2012 Workshop, Held in Conjunction with MICCAI 2012, LNCS vol. 7746, Springer, 2012.

Acknowledgments

Completing this doctoral work has been a great experience. I have received support and encouragement from a great number of individuals and benefited from collaborations with a number of outstanding scientists. I would like to thank Prof. Efthimios Kaxiras for the mentorship, guidance, and support that he has provided over my years in graduate school. He has helped me leverage my interest in large scale computing and apply it to real world problems. Through work with Prof. Kaxiras, I was fortunate to collaborate with the wider Multiscale Hemodynamics Team consisting of physicians from Brigham Women's Hospital, physicists from the École polytechnique fédérale de Lausanne (EPFL) and the National Research Council of Italy (CNR), computer scientists from Argonne National Laboratory (ANL), Lawrence Livermore National Laboratory, and the University of Illinois at Urbana Champaign, and visualization experts at ANL and Harvard University. Being a part of the Kaxiras group has provided me with a broader context for physics research and exposed me to other methods and fields of focus. I have enjoyed getting to know and work with the other students and post-docs, both past and present, over the years.

I have been privileged to also be a member of Prof. Hanspeter Pfister's group. Prof. Pfister has provided ongoing assistance with my research and taught me a great deal about communicating my research. He has fostered an open and collaborative group that has not only exposed me to work in graphics and visualization research but also introduced me to career development and productivity techniques.

I would like to thank everyone from the Harvard Institute of Applied Computa-

tional Science and the Harvard Academic Computing Facility for their support and assistance over the years. I wish to thank Joy Sircar for introducing me to the computational research being done at Harvard and for many beneficial discussions that were both research and career. His excitement about computational research has been an inspiration.

This work was generously supported both by the National Science Foundation and the Department of Energys Computational Science Graduate Fellowship. Throughout the years, the alumni, administration, and other fellows of the CSGF program have provided not only an amazing support system but also a great group of researchers to bounce ideas off and often collaborate with.

I would like to thank my friends and family for their support, advice, and encouragement during my time at Harvard. I would also like to thank everyone who has served as mentors both formally and informally throughout my time at Harvard. In particular, Alan Edelman, Fred Streitz, Nick Bowen, Kirk Jordan, Mary Fernandez, and Linda Zeger. Their assistance has been instrumental to my research and success.

I am grateful to have had access to many computing resources and ongoing computational support. I greatly appreciate all of the help from Dave Singer and Fred Mintzer with the Blue Gene/P system at the IBM T. J. Watson Research Center. I also relied on the Argonne Leadership Computing Facility at Argonne National Laboratory and computing facilities at Lawrence Livermore National Laboratory, both supported by the U.S. Department of Energy.

For Ed

with love and gratitude

List of Figures

List of Tables

1

Introduction

The mechanical motions, which take place in an animal body, are regulated by the same general laws as the motions of inanimate bodies...and it is obvious that the inquiry, in what manner and in what degree, the circulation of the blood depends on the muscular and elastic powers of the heart and of the arteries, supposing the nature of those powers to be known, must become simply a question belonging to the most refined departments of the theory of hydraulics.

– Thomas Young, *1808 Croonian lecture to the Royal Society [22]*

1.1 Overview

Cardiovascular disease (CVD) is still one of the leading causes of death in the western world and is predicted to be the leading cause of death worldwide. CVD caused approximately one of every six deaths in the United States in 2008. It is projected that each year 785,000 Americans will suffer from a new coronary attack and 470,000 will have a recurrent attach [133]. Even more alarming is that in approximately 50%

of these cases, sudden cardiac death is the first manifestation of the disease [172]. Finding indicators of CVD at an early stage is therefore critical for treatment and prevention. Over the last few decades, physicians have linked key properties to the likely development and progression of heart disease but finding methods to identify and track these quantities for individual patients remains an outstanding question.

While the development of CVD depends on genetic predisposition and systemic risk factors such as high blood pressure or diabetes, the localization of the disease that occurs typically at areas of disturbed flow such as at bifurcations and curvatures, cannot be explained simply by these systemic factors that apply equally to the entire arterial geometry. The role of local flow properties in the causation of these localization patterns has long been asserted (c.f. [59], [130]). Even in the early research related to atherosclerosis, it was shown that hemodynamics factors could account for the disparate locations of atherosclerotic lesions in regions subject to identical systemic risk factors. In 1957, Dr. Texon showed that that the incidence and degree of atherosclerotic sites was tied to the fluid dynamics of the blood [153]. Aoki *et al.* further asserted that fluid mechanics has a controlling and inhibiting effect on the development of atherosclerosis and demonstrated a correlation between lesion sites and regions of low wall shear stress [2]. Studying this correlation between flow patterns in patient-specific geometries and the localization of disease quickly gained a lot of interest (e.g. [6], [16], [51]). While factors like oscillatory flow, eddy formation, and boundary layer separation have been topics of interest, one of the main characteristics

studied in this thesis is level of endothelial shear stress (ESS). Evidence has shown that the locations of atherosclerotic lesions are often tied to regions subject to low endothelial shear stress (ESS) ($< 1.0Pa$) ([100], [24], [167], [90]); however, there is currently no method to measure ESS *in vivo* [80].

To address this need, personalized computer simulations that provide accurate and reliable models of blood flows in the human cardiovascular system have become a focus of research over the last few decades. As the velocity profiles and subsequently the shear stress profiles depend strongly on the arterial geometry, assessing the impact of vessel shape on the flow can provide insight into predisposition for diseases like atherosclerosis [163]. Despite the concerted efforts of researchers, building a detailed, realistic model of hemodynamics is still a formidable computational challenge. The simulation must combine the motion of the fluids, the intricate geometry of the blood vessels, continual changes in flow and pressure driven by the heartbeat, and finally the behavior of red and white blood cells and other suspended bodies, such as platelets and lipids [113]. There is a growing literature base of large-scale hemodynamic simulations (c.f. [159], [126], [62], [63], [42], [108],[79]); however, until now the modeling of fluid dynamics through vessels of realistic shapes and sizes for the duration of multiple heartbeats has remained out of reach.

Additionally, as medical imaging modalities have improved to enable data acquisition of patients at resolutions below $0.1mm$, it has become possible to extract patient-specific 3D geometries of the entire coronary arterial tree in a single heartbeat

[100]. The joint use of these imaging techniques and simulation allow for non-invasive screening of a large number of patients for potential coronary disease.

This thesis presents methods and ideas to help address the challenges of modeling blood flow in large regions of patient specific geometries for long time durations. This will be addressed through computational, algorithmic, and physical advances. A foundation will be provided covering the lattice Boltzmann method [144], the fluid dynamics model being leveraged in the work. Building on this mathematical model, new computational techniques are developed to enable a large-scale parallel model including the coupling to physiological levels of red blood cells and requiring the use of 294,912 processors of the IBM Blue Gene/P supercomputer [113]. Additionally, a focused investigation is presented into high order fluid models and associated parallel optimization techniques.

In order to tackle the challenge of reducing the overall time to solution, decomposition in both the temporal and spatial domains will be used to extend the capabilities of next generation systems and enable the simulation of long time intervals. This is important as there is a fundamental limit to the amount of parallelism that can be extracted from traditional spatial scaling. Algorithmic advances necessary to couple these techniques in a stable and accurate formalism will be presented.

As models of multiple cardiac cycles are achieved, the deformational forces acting on the fluid flow through the arteries as the heart expands and contracts need to be accounted for. A novel heuristic for introducing such forces into the lattice Boltzmann

method in a computationally efficient manner will be presented.

1.2 Contributions

Efficient computational codes that can model flow in real systems are essential for understanding many of these complex phenomena. As we prepare for hardware systems that can sustain an exaflop (10^{18} floating-point operations per second (flop/s)) or more, many computational challenges need to be addressed. Such systems will likely involve lower memory to flop/s footprints, greater levels of concurrency due increased processor counts and use of hybrid architectures, and increased strain on I/O resources. The methods used by today's CFD codes today may introduce limitations of both stability and accuracy on next generation systems. The design of massively parallel simulations that can model long time intervals requires the coupling of different scales, mathematical models, and parallelization schemes. To address these challenges of patient specific computer simulations, this thesis makes the following contributions:

Simulating the Entire Heart Circulation System at High Resolution Through a new topology driven graph partitioning method to handle irregular geometries, an efficient molecular dynamics load balancing scheme, and efficient parallel scaling to 294,912 processors of the IBM Blue Gene/P, a simulation of physiologically accurate red blood cell (RBC) count in the coronary arteries retrieved from high-resolution medical imaging, over 300 million RBCs is presented. Complex and

large-scale geometries, such as the one considered here, are rare in the literature (e.g. [36], [7]) . By leveraging large-scale parallel architectures, this work demonstrates a simulation of cardiovascular flow of unprecedented scale in a geometry from real patient data and at physiological hematocrit values for the length of a full heartbeat [113].

Techniques to Enable Efficient Simulations of Higher Order Models

Recent work has shown that higher order approximations of the continuum Boltzmann equation enable not only recovery of the Navier-Stokes hydrodynamics, but also simulations for a wider range of Knudsen numbers, which is especially important in micro- and Nano-scale flows. These higher-order models have significant impact on both the communication and computational complexity of the application. In this thesis, the parallel optimization of such fluid models is investigated on both the IBM Blue Gene/P and Blue Gene/Q architectures. The tradeoffs of methods such as the use of deep halo level ghost cells that, alongside hybrid programming models, reduce the impact of extended models and enable efficient modeling of extreme regimes of computational fluid dynamics were evaluated [115].

Method for Coupling Temporal and Spatial Decomposition

Even with such optimizations, the time to solution needed to complete the simulations can be daunting and spatial decomposition has its limits. For a fixed-size fluid simulation, the efficiency of larger processor counts will saturate when the number of grid points per core becomes too small. To overcome this fundamental strong scaling

limit in space-parallel approaches, a novel version of the lattice Boltzmann method parallelized both spatially and temporally is presented. This method is based on a predictor-corrector scheme combined with mesh refinement to enable the simulation of larger number of time steps. A quantitative analysis of the potential performance impact of this method is also described [116].

Introduction of Deformational Forces

A technique to account for time dependent deformational forces that impact the fluid flow across heartbeats while leaving the mesh representing the geometry static is described. Drawing from previous research on methods to model the expanding universe, these forces are estimated and cast into a kinetic formalism by using a Gauss-Hermite projection procedure. Their impact on endothelial shear stress is evaluated.

1.3 Structure of Thesis

This thesis is structured into ten chapters. Following this preliminary chapter, the mathematical and physical fundamentals are presented. For each main contribution of this thesis, the motivation of the work will be presented alongside any additional background information that has not been previously provided. The contribution will be presented along with a thorough evaluation of the associated experimental results.

The remainder of the dissertation is structured as follows:

Chapter 2: Presents the necessary background regarding the fluid model used in this thesis. An overview of the lattice Boltzmann method is given as well as an introduction to the key characteristics of blood relevant to this work and the definitions utilized in this thesis.

Chapter 3: Provides an overview of the necessary concepts in parallel computing and a brief introduction to the hardware used in this thesis. Related work modeling blood flow using large-scale supercomputers is discussed.

Chapter 4: Describes the methods used to enable the simulation of the full coronary tree included physiologically accurate red blood cell levels on a massively parallel supercomputer. Methods of memory management, irregular domain decomposition, and model coupling will be discussed.

Chapter 5: Discusses extensions to the conventional LBM that allow for accurate modeling of a wider range of fluid regimes. Techniques to improve computational efficiency of parallel implementations of these models are presented.

Chapter 6: Presents an application of the code to real patient data and analyzes the ability to recover data such as pressure gradient in the flow as compared to values measured *in vivo*.

Chapter 7: Describes the method to reduce the overall time to solution through coupling decomposition of the problem in both the spatial and temporal domains. Numerical results for a model problem of laminar flow in a cylinder as well as flow in

patient specific geometries are evaluated.

Chapter 8: Presents a novel method to account for the deformational forces acting on the fluid over the coarse of a heartbeat while leaving the geometry defining mesh static. The impact of these additional forces are evaluated both in a simplified and real patient geometry.

Chapter 9: Outlook on future work.

Chapter 10: Finally, the results, current limitations, and potential future directions will be discussed.

2

Methodology

> *The lattice Boltzmann methods are accurate and robust computational fluid dynamics solvers in the mesoscale, with elegant computational characteristics.*
>
> – Abdel Monim Mohamed Ali Mohamed Hassan Artoli, *2003 Ph.D. Thesis [5]*

In this chapter, the numerical method is reviewed. The theory behind the lattice Boltzmann method is first reviewed followed by a brief discussion of the boundary treatments being used in this work. The lattice Boltzmann method (LBM), the approach discussed in this thesis, is based an algorithm that can efficiently model flow through complex geometries such as those found in the coronary arteries or the aorta. In order to capture the flow patterns accurately and efficiently, it is necessary to use a method that handles complex boundaries well. The LBM is a low-Mach, weakly compressible solver that recovers hydrodynamic behavior in the limit of small Knudsen numbers. Some advantages of the method include and the high level of

scalability achieved on parallel systems (see, for example, [113], [23], [165], [121]).

2.1 Lattice Boltzmann Method

Conventional solutions to problems computational fluid dynamics rely on the incompressible Navier-Stokes equations [152]. An alternative approach was introduced by McNamera and Zanetti [97] and Higuera and Jimenez [72]. While it can be derived as a numerical approximation to the classic Boltzmann equation ([144], [27]), this method historically evolved from Lattice Gas Cellular Automata (LGCA) in which both time and space are discretized and the fluid is viewed as a population of particles that can move in discrete directions between lattice points [52]. In the LGCA model the occupations were modeled by a boolean. This notion was replaced with the distribution function that represented an ensemble of averaged populations moving with each discretized velocity at each lattice point. It was shown that the Navier-Stokes equation could be obtained to a second-order approximation with proper definition of the equilibrium distribution [124].

On general grounds, kinetic theory provides the conceptual framework to bridge micro and macroscales, and the Lattice Boltzmann (LB) method is extremely well suited for the numerical solution. The LB formalism comes from kinetic theory and is a minimal form of the Boltzmann equation based on the collective dynamics of fictitious particles that represent a local ensemble of molecules moving between the points of a regular Cartesian lattice. The dynamics of such particles reproduces

hydrodynamics in the continuum limit, when the molecular mean free path is much shorter than typical macroscopic scales. The fundamental quantity is the particle distribution function, denoted $f_i(\vec{x}, t)$, which represents the probability of finding particles traveling with velocity ξ at lattice node x and at time t. The mesh spacing is defined by Δx, where the discrete velocities $\mathbf{c}_p$ connect mesh points to first and second neighbors. The fluid populations are advanced in a timestep Δt through the following evolution of $f_i(\vec{x}, t)$ with time as:

$$f_i(\vec{x} + \vec{c}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t) - \omega \Delta t [f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t)] \tag{2.1}$$

where $f_i^{eq}(\vec{x}, t)$ is the equilibrium distribution and ω is the dimensionless relaxation parameter (related to the frequency of particle collisions) [144]. In much of this work, the19-speed cubic D3Q19 lattice connecting each lattice point to its first and second neighbors is typically employed unless otherwise cited [92]. In this case, the particles at each lattice point always move along the straight paths defined by the 18 discretized directions along the Cartesian axes and 12 velocities combining two coordinate directions or can stay at rest. Each velocity is assigned a specific weight, ω.

There are two key components to the algorithm: advection and collision. The advection step propagates the fluid particles along the discretized velocity paths defined by the lattice.

Collisions are calculated via a relaxation toward local equilibrium, as illustrated in

the right hand side of Eq.(5.1). The Bhatnagar-Gross-Krook (BGK) collision operator with a single relaxation time scale [17] is leveraged. The local equilibrium is the result of a second-order expansion in the fluid velocity of a local Maxwellian with speed $\vec{u}$ and is defined by:

$$f_i^{eq} = w_i \rho \left[1 + \frac{\vec{c}_i \cdot \vec{u}}{c_s^2} + \frac{1}{2} \left(\frac{(\vec{c}_i \cdot \vec{u})^2}{(c_s^2)^2} - \frac{u^2}{c_s^2} \right) \right] \tag{2.2}$$

where rho denotes the density, $\vec{u}$ the average fluid speed, $c_s = 1/\sqrt{3}$ the speed of sound in the lattice, and w_i the weights attributed to each discretized velocity as determined by the lattice structure. A no-slip boundary condition is enforced at the walls through a full bounce back scheme that will be discussed later. The relaxation frequency ω controls the kinematic viscosity of the fluid:

$$\nu = c_s^2 \Delta t \left(\frac{1}{\omega} - \frac{1}{2} \right) \tag{2.3}$$

The continuum-level fluid viscosity, τ, is defined by Eqn. 2.4.

$$\tau = \nu / c_s^2 + \frac{1}{2} \tag{2.4}$$

Similar to kinetic theory, the macroscopic quantities like density, velocity, and pressure can be calculated through moments of the distribution function which are available locally, with no need of resorting to any expensive Poisson solver. This allows the local mass density to be defined by Eqn. 2.5, the mass current with Eqn. , and the momentum-flux tensor with [101].

$$\rho(\mathbf{x},t) = \sum_i f_i(\mathbf{x},t) \tag{2.5}$$

$$\rho\mathbf{u}(\mathbf{x},t) = \sum_i \mathbf{c}_i f_i(\mathbf{x},t) \tag{2.6}$$

$$\overleftrightarrow{P}(\overrightarrow{x},t) = \sum_i f_i(\overrightarrow{x},t)\overrightarrow{c_i}\overrightarrow{c_i} \tag{2.7}$$

One drawback is that the LBM requires many small time steps as limited by CFL-type conditions. These time steps are extremely efficient and make the method amenable for parallel implementations [115]. However, the most compelling asset of LB rests with its outstanding amenability to parallel computing, *even in complex geometries*. The present work represents a major testimonial to this asset, demonstrating excellent scalability on up to $\sim 300K$ cores on a real-world patient geometry. The endothelial shear stress (ESS) is calculated via the tensor second invariant in Eq. 2.10, in which $\overrightarrow{x_\omega}$ denotes the the position of the sampling points near to the mesh wall nodes [19].

The wall shear stress can be calculated using the shear tensor defined by Eq. 2.8 and evaluated though the kinetic representation in Eq. 2.9.

$$\overleftrightarrow{\sigma}(\overrightarrow{x},t) = \nu\rho(\overrightarrow{\partial x}\overrightarrow{u} + \overrightarrow{\partial x}\overrightarrow{u^T}) \tag{2.8}$$

$$\overleftrightarrow{\sigma}(\overrightarrow{x},t) = \frac{\nu\omega}{c_s^2}\sum_i \overleftrightarrow{c_i}\overleftrightarrow{c_i}(f_i - f_i^{eq})(\overrightarrow{x},t) \tag{2.9}$$

$$S = (\overrightarrow{x_\omega},t) = \sqrt{(\overleftrightarrow{\sigma} : \overleftrightarrow{\sigma})(\overrightarrow{x_\omega},t)} \tag{2.10}$$

In this equation, x_ω denotes the position of the geometric wall, or sampling points in close proximity to the mesh walls. S provides a direct measure of the degree of shear stress exhibited near the wall [100].

2.2 Boundary Treatments

The LB formalism provides a handier treatment of complex geometries such as those seen in real world cardiovascular problems. This comes at the expense of numerical accuracy, which usually degrades to first order, due to the staircase representation of arbitrarily shaped boundaries. This weakness is, however, strongly mitigated by two compensating effects. First, the shear stress is available *locally*, as a linear combination of the discrete populations sitting at each given cell, as shown by Eq. 2.10, which relieves the burden of computing spatial derivatives of the velocity field at the boundaries, which is an accuracy-threatening procedure used in Navier-Stokes solvers. Second, while staircase boundaries do compromise second-order accuracy, it is also true that a favorable pre-factor (due again to the straight discretized velocity trajectories) secures significant error reduction by increasing resolution of the fluid model.

In practice, wall shear stress is found to converge to acceptable levels of accuracy at grid resolutions below 20 microns [101].

A no-slip boundary condition is imposed at the wall through the use of a full bounce-back method. To this end, the velocity of any particle, which is set to advect to a lattice point designated as a wall node, is reversed. In this case, the directions of post-collisional particles are reversed if the prescribed velocity points to a lattice point designated as a wall node as shown in Fig. 2.1. The curved vessels are shaped on the regular (axis-aligned) grid via a staircase representation as opposed to the body-fitted grids found in direct Navier-Stokes solvers. This does come at the expense of numerical accuracy, which has been shown to degrade to first order [144]. This representation is improved systematically by increasing the resolution of the mesh via increased density of lattice points.

This consists of reversing at every time step the post-collisional populations towards a wall node, providing first-order accuracy for irregular walls.

The method of prescribing the inflow and outflow rates within the different simulations that are discussed in this thesis varies widely. To that end, these detail implementations will be discussed for each specific case.

A constant velocity is imposed at the inlet through a *plug profile* at the entrance to the vessel. While this does not assert the known parabolic profile that drops to zero close to the wall, it allow a total flow to be imposed at a set value. In a short distance past the inlet, the parabolic profile is recovered. At the outlets, a constant

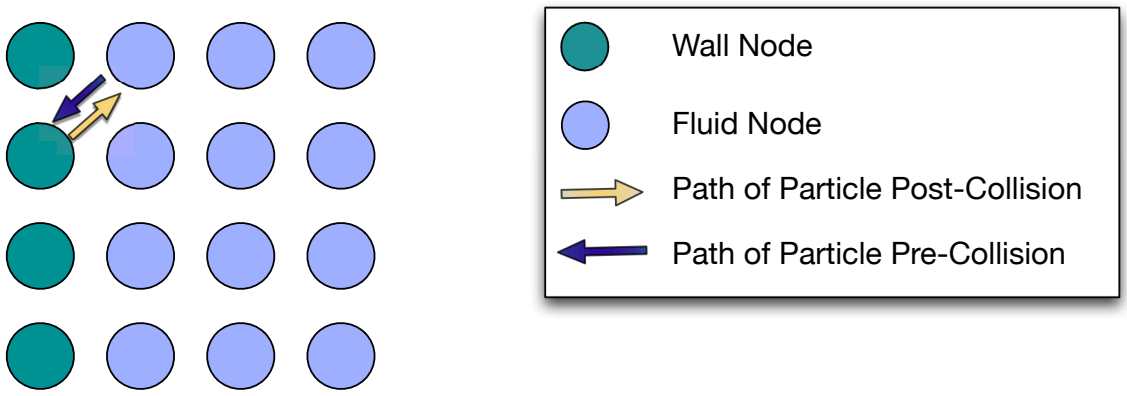


Figure 2.1: Bounceback boundary condition. Each particle reverses direction as it encounters a wall node. The green circles indicate the wall nodes. The blue circles show the fluid nodes. The yellow and dark blue arrows indicate the path of post- and pre-collision respectively.

pressure is imposed allowing pressure to be built up between the inlets and outlets. The Zou-He boundary conditions [173] are used to implement these conditions. This method uses information streamed from the bulk fluid nodes alongside a completion scheme for the unknown particle populations whose neighbors are outside the fluid domain. This method can be executed with second-order accuracy [89]. In this thesis, the modification introduced by Hecht and Harting [69] is used in which the velocity conditions are specified on-site thus removing the constraint that all nodes of a given inlet or outlet must be aligned on a plane that is perpendicular to one of the three main axis. Furthermore, this addition allows the boundary conditions to be applied locally. Pulsatile hemodynamics requires the investigation of the flow pattern in the time domain. It's simulated via a time-dependent influx derived from physiological data and discussed in more detail in Chapter 6.

2.3 Hemodynamic Specific Parameters

The circulatory system is a complex maze of vessels designed to facilitate, control, and maintain blood flow to all parts of the body. Blood is a concentrated suspension of particulates primarily including erythrocytes (red blood cells or RBCs), leukocytes (white blood cells or WBCs), and platelets [147]. These components are suspended in a plasma that is generally regarded as a Newtonian fluid [147]. One microliter of human blood contains about 5,000,000 red blood cells, 7,000 white blood cells, and 300,000 platelets [86]. As the majority of the particle volume is occupied by the erythrocytes (99%), hematocrit is defined as the proportion of blood volume that is occupied by erythrocytes. For humans, it has been experimentally shown that hematocrit is equal to 47 $\pm$5% for healthy adult males and $42 \pm 5\%$ for healthy adult females ([166], [87]). These cells dramatically influence the behavior of the fluid. The resulting simulation specific parameters are discussed in the following sections.

2.3.1 Viscosity

Dynamic viscosity, ν, is one of the most significant mechanical properties of blood. Viscosity defines the tendency of a fluid to resist flow which relates to the shear rate, γ and the shear stress S. When $S = -\nu\gamma$ then the viscosity is independent of the shear rate and defined as a Newtonian fluid. The kinematic viscosity ν is defined as the ratio of the dynamics viscosity to the density, $v = \nu/\rho$.

The viscosity of blood is further dictated by the viscosity of the blood plasma and

the density and nature of the suspended particulates. Blood plasma shows viscosity, while whole blood is both viscous and elastic. Patients suffering from anemia will have a lower hematocrit level and subsequently a lower viscosity while conversely, those exhibiting polycythemia will have blood of a higher viscosity and RBC count. It is also worth noting that the deformability of the cells can impact the viscosity of blood and introduce elastic properties. The elastic behavior is derived from the energy due to the movement and deformation of red blood cells [155]. For patient's with more rigid cells, the viscosity of the blood increases. This can be of significant interest for patients that suffer from sickled cell anemia [41]. However, typically the effects of variation in blood viscosity are exhibited significantly typically in the microcirculation and not evident in the large arteries geometries considered in this thesis. At low shear rate, below 10 sec^{-1}, the cells are heavily clustered and the blood is categorized as non-Newtonian. For medium shear rates between ten and 100 sec^{-1}, the clusters dissolved and the cells start to orient themselves. The viscosity decreases as the shear rate increases. Finally, in regimes of large shear rates above 100 $sec-1$, the blood can be fairly approximated as Newtonian [5].

A change in temperature can change the viscosity as well. For instance, a decrease of only 1° Celsius can result in a 2% increase in viscosity. In this thesis, the temperature is taken to be 37° Celsius and the hematocrit level to be 45% unless otherwise noted. This corresponds to a viscosity of $3.2mPa \cdot s$ [163].

2.3.2 Density

The density of blood is closely tied to the hematocrit level, or proportion of blood volume occupied by red blood cells, and can vary from patient to patient. In this thesis, density value of 1.06 $\frac{g}{cm^3}$ is used [71].

2.3.3 Reynolds Number

The Reynolds number (Re) refers to the ratio of inertial force to viscous force and is defined as $u\rho l/\nu$, where u is the mean blood velocity, ρ the fluid density, ν the kinematic viscosity of blood, and l the diameter of the vessel [143]. In the case of low Reynolds numbers, the viscous forces dominate while the inertial forces dominate systems with high Reynolds numbers. The Reynolds number works as an indicator of how close the fluid field is to a turbulent regime. In terms of the coronary arteries, the Reynolds numbers have been shown to range from 100-460 [80]. These values fall well below the 2000 cutoff for producing and maintaining turbulent flow. Moreover, experimental data has demonstrated that the blood flow will remain laminar in these regimes [143].

For large arteries like the aorta, the Reynolds number is 1150 when assuming rigid walls, but ranges only from 1035-1265 when taking into account wall pliability. This shows the diminished impact of wall flexibility in large vessel models and allows the conclusion that, to the first approximation, imposing rigid walls is a valid assumption for large arteries [5]. In this thesis, the walls will be treated as rigid.

2.3.4 Mach Number

The Mach number (Ma) helps to assess the fluid's compressibility by measuring the fluid's velocity relative to the speed of sound in the fluid. It is defined as $M = u/c_s$ where u is again the average fluid velocity and c_s is the speed of sound. It is significant here simply due to the semi compressible nature of the selected lattice Boltzmann method and helps to ensure the validity of the model [60].

3

Parallel Computing

> *For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution.*
>
> – Gene Amdahl, *AFIPS 1967 Conference [1]*

Parallel programming is a technique used to spread computation across a number of processors in an effort to reduce the overall runtime of an application and to maximize the use of the available hardware. In this chapter, a brief introduction to key concepts in parallel computing and its application to cardiovascular hemodynamics is provided.

3.1 Overview

The goal of parallelizing an application is to harness the power of multiple cores and derive a subsequent speedup in the overall runtime as more cores are utilized. Speedup here refers to how much faster the parallel implementation is to the sequential

implementation and can be calculated with Eq. 3.1 in which S refers to the speedup, Ts to the sequential time, and Tp to the duration of the parallel algorithm.

$$S = \frac{Ts}{Tp} \tag{3.1}$$

Ideal or linear speedup occurs when $S = N$, N is the number of processors. Essentially, if two processors are used then in the ideal case one would achieve a $2\times$ speedup. Similarly, if 100 processors were used, one would wish to achieve a $100\times$ speedup. In the simplest case, an algorithm can be broken down into many serial chunks that require little to no communication. If it can be broken down into this set of independent tasks, the algorithm is described as *embarrassingly parallel* or *pleasantly parallel*. In the idealized case, if there were no overhead to the run, using N cores would result in an $N - \times$ speedup.

A related quantity of interest in this thesis is the notion of *parallel efficiency*, Ep, that defines how close the parallel scheme comes to the ideal. It can be calculated by $\frac{s}{N}$ and is often denoted as a percent. For example, if 100 processors are used and an $80\times$ speedup is achieved, one would say the implementation has a parallel efficiency of 80%.

As mentioned, in an ideal situation one would hope to see a direct match between the number of processors and the speedup achieved; however, this is typically not achievable due to factors like communication overhead and sections of the code that are inherently serial. It is therefore important to get a sense of how well a particular algorithm can be mapped to a parallel architecture. The first step to understanding

potential speedup is to examine the code and identify any existing interdependencies. In 1967, Gene Amdahl offered a basic model to understand potential gains for parallelizing an application that has come to be known as Amdahl's Law [1]. He proposed a method of defining S through Eq. 3.2 to identify the degree to which the speedup of the application is limited by the serial portion of the code.

$$S = \frac{1}{(1-P) + \frac{P}{N}} \tag{3.2}$$

In this case, P represents the proportion of the code that can be parallelized and N represents the number of processors. In the case of fluid dynamics models using lattice Boltzmann, the stream and collide function being handled for one specific discrete velocity at one grid point in one time step is the smallest section of serial code that the algorithm can be broken down into. The time steps must execute in lock step thereby introducing a time dependence and inherent serial characteristic. Amdahl's law shows that the speedup achieved for this application will never be better than that proportion of serial code (1-P) or in the case of the LBM, the time for each stream and collide execution.

This model still provides an overly optimistic view of potential gains through a parallel implementation. As stated above, the parallel portion of the code would continue to decrease at the same rate per new processor added. This is referred to as *strong scaling* and demonstrates a reduction in runtime as the problem space is held fixed and the processor count is increased. Simply relying on Amdahl's law

can be misleading. A straightforward example is useful to provide insight into these drawbacks. Many embarrassingly parallel codes utilize a scheme known as the master/worker setup. In these applications, one processor acts as the *master* distributing work units to the other *slave* processors. In accordance with Amdahl's Law, as the core count increases there are subsequently more slaves available to execute the work units in parallel. The issue arises when all work units have been distributed. Clearly at this point, any additional processors will sit idle and not contribute to any further speedup. Additionally, Eq. 3.2 does not take into account factors like messaging overhead.

In 1988, John Gustafson proposed a law to address this first issue by removing the fixed problem size constraint and instead having the problem size increase as the machine size increases. This is significant as the goal for parallelizing many applications is often not simply to reduce the overall runtime, but to enable the study of much larger problem sizes. Often times, the point of growing the system is to enable this type of large-scale science that would otherwise be infeasible. This is referred to as *weak scaling*. The method that is now referred to as Gustafson's Law provides a modified equation for speedup as demonstrated in Eq. 3.3 [66]:

$$S = N - \alpha(N - 1) \tag{3.3}$$

In this case, N is again the number of processors and α is the serial portion of the application. This address some of the points where Amdahl's Law falls short and

provides a slightly more robust expectation for a parallel version of the code. For the algorithm discussed in this thesis, this is extremely significant as a parallel implementation will enable the problem size to be increased drastically. When modeling flow in the coronary arteries, it is important to model both larger fluid systems and longer time durations.

The two laws described above still leave a lot to be desired in the prediction of actual speedup to be obtained by a parallel code. These analyses both neglect significant bottlenecks like memory and I/O bandwidth. The rate at which these scale with the number of processors will have an impact on the exhibited speedup. In many cases an algorithm–such as those described as embarrassingly parallel–may theoretically scale linearly with the system size, but communication overhead will cause it to plateau or taper off in practice. Another concern is the method of coding the application. For example, if the previously discussed master/worker scheme is employed, the applications often have the master handle all of the I/O in the beginning and end of the application. This not only results in high communication volume to distribute the data and retrieve results, but often requires a large memory footprint for the master to be able to handle any post-processing or data collection. This scheme often requires special care to be taken in memory management in order to achieve stronger performance. As applications scale to large system sizes, these issues of memory management, communication overhead, and I/O become key factors determining achieved speedup. How an algorithm is affected by these factors can help determine the best

architecture for the code to be ported to. If the application requires a large amount of inter-processor communication, hardware designed with optimized networks would be more ideal whereas an application with completely independent chunks may require hardware designed for memory management.

In the following chapters, refined equations for speedup to address concerns particular to the implementation of the LBM presented will be presented alongside data demonstrated a strong correspondence between the theoretical and achieved speedup.

3.2 Architecture

For the work described in this thesis, the architectures used come from the family of IBM Blue Gene supercomputers (c.f. [53], [74],[67]). All three supercomputers, Blue Gene/L (BG/L), Blue Gene/P (BG/P), and Blue Gene/Q (BG/Q), were used over the coarse of the presented research. The architecture for these systems is based on low cost embedded PowerPC technology and is based on the notion of achieving large performance at low power by coupling a large number of low power processors together. Some of the architectural features are relevant for this thesis, so I will briefly summarize some of the key components in this section.

The Blue Gene system is a massively parallel supercomputer that uses a distributed memory setup. The basic building block is a custom system-on-a-chip (SoC) that integrates processors, memory, and communication. All three chips consist of low frequency processors running at 700 MHz (BG/L), 850 MHz (BG/P), and 1.6 GHz

(BG/Q). Each compute node contains 2, 4, and 16 cores respectively and exhibit a peak performance of 5.6 GFlop/s, 13.6 GFlop/s, and 204.8 GFlop/s. Partitions of varying sizes include compute nodes and I/O nodes. The ratio of I/O nodes to compute can vary and depending on how data intensive your application is, this can impact the performance.

Four highly optimized networks connect the nodes: a three-dimensional Torus (five-dimensional in the case of Blue Gene/Q), Global Collective Network, Gigabit Ethernet networks, and Control System Network. The majority of messaging is conducted via the torus network that supports low-latency, high bandwidth point-to-point messaging. The specialized networks are a key attribute of this system. They were specially designed with message passing applications in mind such as those relying on the standard MPI protocol. Generally one of the largest sinks for parallel code comes from communication overhead. This fine-tuned arrangement allows for efficient point-to-point communication along the torus and multiple node collective communication across the global collective network. The torus interconnects all compute nodes and the Global Collective Network provides both broadcast and reduce functionalities between all nodes. The latency of the tree traversal is on the order of microseconds. This collective network allows for standard MPI calls like Broadcast and AlltoAll to be complete in only a few clock cycles. As the Blue Gene system scales to such a high number of processors, the ability to communicate succinctly between the cores is a key factor when scaling applications. For more information

regarding the networks refer to [104].

Another quantity of interest is the *hardware efficiency* which refers to the amount of available system performance that a particular application can leverage. Typically this is measured in terms of achieved floating-point operations per second (flop/s) over the theoretical peak flop/s that can be achieved by the system. In this thesis, the Hardware Performance Monitor from the IBM HPC Toolkit is used to measure hardware efficiency [33].

3.3 Related Work

Modeling fluid dynamics of a biological nature has been an area of interest for many years (c.f. [158], [84]). Specifically applying computational methodologies to identify regions prone to cardiovascular disease dates back to the work by DeBakey [32] and Thubriker and Robicsek [154]. Since then, research has typically focused on either accurate modeling of the red blood cells and other particulate components of the blood or on the fluid dynamics of blood moving through patient representative geometries. Many models for red blood cells have been explored such as the immersed boundary method ([112], [38]), dissipative particle dynamics ([161], [109], [73]), or linear finite element analysis (FEA) which has been shown to efficiently model hundreds of cells ([88], [170]).

Alternatively, a great deal of research has focused on understanding the underlying mechanisms that experimental measurements alone could not have achieved. A

recent study demonstrated the use of computational methods to guide cardiovascular intervention. In this case, the focus was on determining fractional flow reserve (FFR) to assess the significance of coronary lesions [78]. Alongside this there has been a great deal of research applying large scale supercomputing to model hemodynamics in patient specific geometries (e.g. [150], [160], [5], [60], [63], [64],[30], [79]). Much of this work focuses on flow in small regions of arteries and only in 2012 were the some of the first simulations of flow in full body large arterial networks presented [168].

The disease trajectories that can be modeled, however, have been limited by the rate at which these simulations can be performed. A significant challenge is to capture, *in silico*, functionally important biological events that typically occur on the timescales of anywhere from seconds to decades. Furthermore, an efficient method of modeling physiological red blood cell levels and coupling to the fluid component is required. Throughout the following chapters, previous work related specifically to the contribution being presented will be provided.

4

Parallel Implementation and Scaling

The issue of coupling models of different events at different scales and governed by different physical laws is largely wide open and represents an enormously challenging area for future research.

– Brown *et al.*, *U.S. Department of Energy Report [20]*

4.1 Motivation

As mentioned, accurate and reliable modeling of blood flows in the human cardiovascular system has the potential to improve understanding of cardiovascular diseases, which are the most common cause of death in Western countries. But building a detailed, realistic model of hemodynamics is a formidable computational challenge. The simulation must combine the motion of the fluids, the intricate geometry of the blood

vessels, continual changes in flow and pressure driven by the heartbeat, and finally the behavior of red and white blood cells and other suspended bodies, such as platelets and lipids.

In this chapter, the first multiscale simulation of cardiovascular flows in realistic human arterial geometries derived from Computed Tomography Angiography (CTA) data is presented. The simulation covers the entire heart circulation system, the network of arteries and arterioles that supply blood to the heart muscle, with a spatial resolution extending from 5 cm down to $10\,\mu$m.

The simulations involve up to a billion fluid nodes, embedded in a bounding space of about a three hundred billion voxels, with 10-300 million suspended bodies. They are performed with the multiphysics code MUPHY (MUlti PHYsics/multiscale), which couples Lattice Boltzmann methods for the fluid flow and a Molecular Dynamics treatment of the suspended bodies [13, 100]. The simulation achieves an aggregate performance in excess of 60 teraflops, with a parallel efficiency of more than 60 percent on a full $294,912$-processor BlueGene/P configuration.

This work presents a number of unique features, both at the level of high-performance computing technology and in terms of physical/computational modeling. The extremely complicated conditions that are implicit to irregular geometries require that the workload be evenly distributed across the pool of as many as $294,912$ computational nodes of the BlueGene/P supercomputer. The formidable graph-partitioning problem, even at the mere level of the fluid computation, cannot be overestimated. On

top of this, our multiphysics/scale application adds the further constraint of keeping a good workload balance also across the Molecular Dynamics (MD) sector of the simulation. To the best of our knowledge, the latter issue has never been tackled before in any MD simulation. Indeed, even top-ranking (Gordon-Bell winning) multi-billion MD simulations are invariably performed in idealized geometries, cubes or regular boxes [118]. Similarly, multi-billion node simulations of, say, biofluid turbulence are indeed available, but only in the same ideal geometries mentioned above. Complex and large-scale geometries, such as the one considered here, are rare in the literature (c.f. [36], [7], [61]). By leveraging large-scale parallel architectures, this work demonstrates a simulation of cardiovascular flow of unprecedented scale in a geometry from real patient data and with blood flow at physiological hematocrit values. In this paper, the treatment of red blood cells as extended structures (i.e. not as point sources), a method to handle highly irregular geometries via topology driven graph partitioning, and an efficient MD load balancing scheme are introduced.

4.2 Multiscale hemodynamics

The approach is based on efficient and accurate algorithms capable of handling the requirements of the diverse computational entities and associated scales. The numerical framework is handled by the software *MUPHY* developed by our group in recent years [13]. The approach is genuinely multiphysics, as it combines different levels of the description of matter, continuum hydrokinetic fluids for the dynamics of

blood plasma and individual particles for the representation of red blood cells and other minority suspended species. The method is also multiscale, since fluid and particles are advanced concurrently and the exchange of information is computed on-the-fly.

Finally, the last term in Eq. (5.1) represents the coupling between fluid and suspended bodies. This is given by

$$\begin{aligned} \Delta f_p(\mathbf{x},t) &= -w_p \Delta t \left[\frac{\mathbf{G}\cdot\mathbf{c}_p}{c_s^2} + \right. \\ &\left. \frac{(\mathbf{G}\cdot\mathbf{c}_p)(\mathbf{u}\cdot\mathbf{c}_p) - c_s^2 \mathbf{G}\cdot\mathbf{u}}{c_s^4} \right] \end{aligned} \tag{4.1}$$

where $\mathbf{G}$ is a forcing term containing the translational and rotational exchange of momentum induced by N moving red blood cells at position $\{\mathbf{R}\}$. The forcing term is smeared over a region made of 32 mesh points around each RBC and with ellipsoidal shape. The drag force acting on particles is modelled as

$$\mathbf{F}_i^D(\mathbf{R}_i) = -\gamma^T (\mathbf{V}_i - \tilde{\mathbf{u}}) \tag{4.2}$$

and the torque is

$$\mathbf{T}_i^D(\mathbf{R}_i) = -\gamma^R \left(\mathbf{\Omega}_i - \tilde{\mathbf{\Omega}} \right) \tag{4.3}$$

with $\{\mathbf{V}_i\}$ and $\{\mathbf{\Omega}_i\}$ being the RBC velocities and angular velocities, and with $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{\Omega}}$ the fluid velocity and vorticity fields, smeared over the same ellipsoidal region occupied by a RBC. This smearing is achieved through an envelope function similar to the one used in the Immersed Boundary method [112], which takes into account the finite extent of the particles by means of a smooth interaction. The constants γ^T

and γ^R are translational and rotational coupling coefficients of RBCs represented as oblate ellipsoids in a hydrodynamic environment.

Due to the finite extent representation of an RBC, the hydrodynamic size of RBC is smaller than the smearing region covered by the particle, with the RBC effective size depending on the strength of the coupling coefficients γ^R and γ^T. By varying these coefficients, and matching the hydrodynamic volume with the GB exclusion volume, the effective extension of a RBC covers ~ 1 lattice cell. This corresponds to a hematocrit level of 1%. To reach a more physiological level of $30 - 45\%$ about 300 million RBCs are required.

Here a compromise between physical fidelity and computational efficiency must be taken. Indeed, recent studies [119] indicate that the minimum number of degrees of freedom required for a quantitative description of RBC dynamics in a fluid flow, including deformability, is of the order of hundreds to thousands. This is far too much for a viable fluid-particle coupling at large-scales. As a result, an intermediate strategy has been developed, whereby RBCs are treated as rigid ellipsoidal bodies (six degrees of freedom) interacting with each other through custom potentials, and with the surrounding fluid (the blood plasma) via tensorial mobility coefficients, accounting for the anisotropic drag experienced by the RBC along and across the local fluid direction of motion. Such an intermediate strategy permits to capture the essential features of the complex behavior of the RBCs and their impact on the macroscopic blood rheology, at a very moderate computational cost. The represented behavior

accounts not only for the translational and rotational motion of the RBCs, but also for their mutual interaction, reproducing aggregation patterns of RBCs and their impact on the overall behavior of the blood flow.

From an algorithmic point of view, this approach scales linearly with the number of RBCs, thanks to the fact that the solvent-mediated RBC-RBC interactions are entirely local and explicit. This stands in marked contrast with consolidated approaches, based on Brownian dynamics, which rely upon a non-local Green function representation of the Oseen tensor and consequently can only attain $N \log N$ scaling with the number of RBC's by resorting to highly sophisticated procedures. The strategy described here, which is entirely new and still unpublished [98], makes therefore a particularly efficient use of the invested computational resources. In particular, as typical of LB applications, it provides a very economical algorithmic representation of fairly complex physical phenomena. In the present implementation, additional torques arising from coupling with the elongational component of the flow pattern and tank treading of the RBCs are neglected.

Mechanical hard core forces prevent contacts between RBCs. The RBC-RBC interactions are pairwise and modelled via the Gay-Berne potential [55], reading

$$u_{ij}^{GB}(q_{ij}) = 4\epsilon(q_{ij}) \times \left[\left(\frac{\sigma_0}{R_{ij} - \sigma(q_{ij}) + \sigma_0} \right)^{12} - \left(\frac{\sigma_0}{R_{ij} - \sigma(q_{ij}) + \sigma_0} \right)^{6} \right] \tag{4.4}$$

where $q_{ij} \equiv (R_{ij}, \hat{\mathbf{u}}_i, \hat{\mathbf{u}}_j)$ and with R_{ij} being the relative distance, $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_j$ are

the principal directions of the i-th and j-th ellipsoids, with $\epsilon(q_{ij})$ and $\sigma(q_{ij})$ being functions with lengthy expressions reported in ref. [55].

The potential u_{ij}^{GB} is set to zero beyond a orientation-dependent cut-off given by the condition

$$\left(\frac{\sigma_0}{R_{ij} - \sigma(q_{ij}) + \sigma_0}\right)^6 > 0 \tag{4.5}$$

to retain the repulsive component of the potential only. The rigid body dynamics of the suspended bodies is propagated in time via a second-order accurate timestepping algorithm [35], properly modified to handle fluid-particle forces and torques.

4.3 Geometry acquisition and mesh-generation

The global geometry of the problem used for the present simulations is obtained from CTA scans of the coronary arterial system of a real patient. Data acquisition was performed by a 320×0.5 mm CTA scanner (Toshiba) and subsequently segmented into a stack of two-dimensional contours at a nominal resolution of 0.5 mm. The slice contours, each consisting of 256 points, are oversampled along the axial distance down to a slice-to-slice separation of 12.5 μm and further smoothed out by appropriate interpolators. The resulting multi-branched geometrical structure is finally mapped into the Cartesian LB lattice, ready for the simulation. Full details can be found in [100].

4.4 Initial and Boundary conditions

Fluid boundary conditions are set up as follows. At the inlet, a uniform flow profile with prescribed velocity is imposed, and at the outlet ports a zero pressure difference from the inlet is maintained. The flow-pressure inflow/outflow conditions are implemented via the Zou-He method to set up the LB populations in the proper way [173]. At rigid walls, a standard mid-way bounce-back rule is applied to impose no-slip flow conditions.

The fluid flow is initialized with zero speed and constant density across the entire domain. Particles are seeded at random positions and orientations, and with null linear and angular velocity. In flow conditions, RBC that exit from the outlet ports are reinjected in the inlet port in order to maintain a constant total hematocrit. The injected RBC have velocity given by the imposed inlet velocity, random orientation and zero angular velocity. The RBCs are repelled by the wall via a GB pairwise potential acting between a RBC ellipsoid and a spherical particle positioned on a wall mesh node.

4.5 Code Features

The *MUPHY* (MUlti PHYsics/multiscale) code is written in Fortran 90 and uses MPI for the parallelization. To handle in a flexible and efficient way any complex geometry, *MUPHY* makes use of an indirect addressing scheme that has been described

along with other main features of the code in [13]. We showed in the same paper that the penalty introduced by the indirect addressing scheme for the cases of regular geometries is very limited ($\sim 5\%$ of the execution time) and, as a matter of fact, the code was used for problems, such as bio-polymer translocation in which the geometry is trivial (a regular box). Originally developed for the IBM BlueGene/L system [54] *MUPHY* has been recently ported to heterogeneous clusters of CPUs and Graphics Processing Units (GPU), using the CUDA software environment, showing excellent results [12]. For the present work the latest generation of the IBM Bluegene system is employed whose main features may be summarized as follows:

- quad SMP processor chip per node with 2GB of memory per node.
- system-on-a-chip design with superscalar 850 MHz PowerPC 440 cores;
- a large number of cores (scalable up to at least 294,912);
- three-dimensional torus interconnect with auxiliary networks for global communications, I/O, and management;
- lightweight, Unix-like OS per node for minimum system overhead.

The first versions of the LB component of the code used the "fusion" of the collision and streaming steps in a single loop. This technique, by now standard in all high-performance LB codes, significantly reduces data traffic between main memory and cache/registers of the processor, since there is only one read and one store of

all LB populations at each time step. However most implementations of the fused update resort to a "double buffer" to store the LB populations. The double buffer avoids the mixing of old and new data during the non-local streaming step that would be a source of inconsistency but, obviously, doubles the memory required for the LB populations. A recent enhancement to *MUPHY* is the implementation of the "single buffer" mechanism for the Lattice Boltzmann update through an adaptation of the so-called swap algorithm [95]. In this algorithm, the particle populations are rearranged after collision. This results in a memory layout in which, as a given population is copied to a neighboring cell during the streaming step, it simply exchanges its memory location with a neighbor-node population. Thus, the copy operations in the streaming step are replaced by exchange operations, or variable swaps, as suggested by the name of the algorithm. Given that no information is lost during the swap operation, the algorithm handles the streaming phase without the need for temporary buffers. This point becomes clear by looking at Fig. 4.1, where four nodes of a one-dimensional Lattice Boltzmann mesh, with just two populations per node, perform a single global collision-streaming cycle, carrying them from a discrete time step t to $t + \Delta t$. Right after the mesh populations have collided, the algorithm rearranges the data locally by exchanging the two populations (in a higher-dimensional case, all local populations are exchanged with the population corresponding to the opposite direction), as indicated by the keyword "swap" on the figure. Then, as soon as the neighboring node also reaches its post-collision state, a non-local exchange operation,

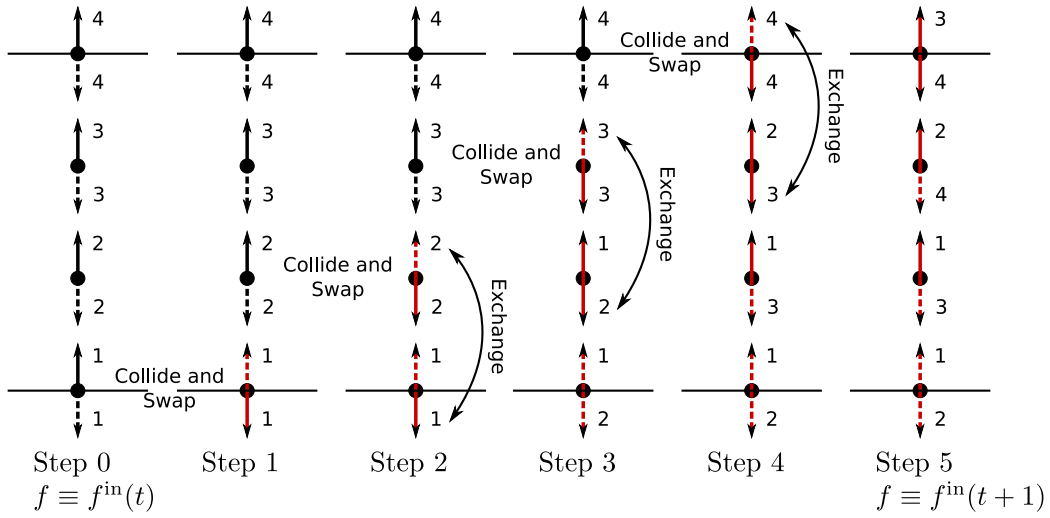


Figure 4.1: Schematic representation of a single collision-streaming cycle on four cells of a one-dimensional Lattice Boltzmann simulation with two populations per cell (detailed explanations are found in the text). The two populations on each cell are distinguished by the use of a solid line for the first and a dashed line for the second. The numbers next to the populations label the cell on which the populations were located at the initial time step t. Red denotes a population that has reached the post-collisional state.

indicated by the keyword "exchange", is performed in lieu of the streaming step.

In a typical execution of a *MUPHY* program, memory is used mainly for the storage of the particle populations and the connectivity list. Thus the swap algorithm reduces the overall memory needs by one-third by avoiding a duplication of memory for the particle populations. Furthermore, this approach leads to a sensible performance improvement, because the program becomes more cache efficient as it holds the populations and connectivity matrix in a smaller memory space.

The geometry gathered from the CTA data that was used in the runs reported in section 7.9 is highly irregular, as shown in Figure 4.4, and its partitioning among the

available processors represents a major challenge in itself. Several domain decomposition strategies for irregular lattices already exist, as described in [96]. In particular, state-of-the-art techniques like those represented by multilevel $k-$way partitioning schemes can be used for irregular geometries. However, when either the size of the mesh or the number of partitions increases to critical values (in our case, the figures are $\sim$ 1 billion nodes for the mesh and $\sim$ 300,000 partitions) most of the widely used tools simply fail, meaning that they are unable to manage the problem (that is much worse, of course, than producing a sub-optimal solution). For instance, the well-known tool for graph-partitioning *METIS* [75], even in its parallel version, requires the allocation on each processor of a block of memory equal in size to the square of the number of partitions. On the other hand, preliminary tests showed that a naive partitioning based only on a balanced number of mesh nodes on each processor produced a very poor load balancing.

It should be emphasized that the graph-partitioning strategy is entirely topology-driven, i.e. it proceeds based on the input provided by the local connectivity supplied by the Lattice Boltzmann grid (18 neighbors, uniformly across the entire computational domain) with no information on the global geometry of the problem.

Finally, an effective solution was found by using *PT-SCOTCH*, the parallel version of the *SCOTCH* graph/mesh partitioning tool [29]. One of the interesting features of *SCOTCH* is that its running time is linear in the number of edges of the source graph, and logarithmic in the number of vertices of the target graph for mapping compu-

tations. Moreover, a test carried out on a smaller case ($\sim 20,000,000$ mesh nodes partitioned among $1,024$ processors) showed that *SCOTCH* produces a partitioning scheme superior to *METIS*, that is, with a better load balancing taking into account both the number of mesh nodes per processor and the total communication among the processors. Unfortunately, *PT-SCOTCH* runs out of memory on our real test case that produces a graph with almost one billion vertices and $\sim$ 18 billion edges. Our solution has been to use a *pruned* graph which represents the connectivity along the six main directions only $(+x, -x, +y, -y, +z, -z)$. This reduces the number of edges in the graph by 66% (by eliminating the 12 edges: $+x+y, +x-y$, *etc*). We confirm that, in a smaller test case, the resulting partition is, for all practical purposes, very similar to the partition produced for the whole graph. By using the pruned graph, the required partition ($294,912$ domains) is created for the large test case on a cluster using 128 Intel cores (Xeon E5520 @ 2.27 Ghz) with a total of 256 GB of memory.

It is interesting to look at the distribution of the tasks with respect to the number of other tasks with which they are required to exchange data, following the partitioning scheme produced by *SCOTCH*. The result is reported in Figure 4.2, which shows that, on average, each task exchanges data with other 15 tasks.

This workload distribution indicated that despite the highly complex geometry, the final workload partitioning ends up relatively close to the initial topological input, which is expected indeed as a heuristic measure of good balance. Visual inspection of the computational domains, shows that this is realized through a fairly sophisticated

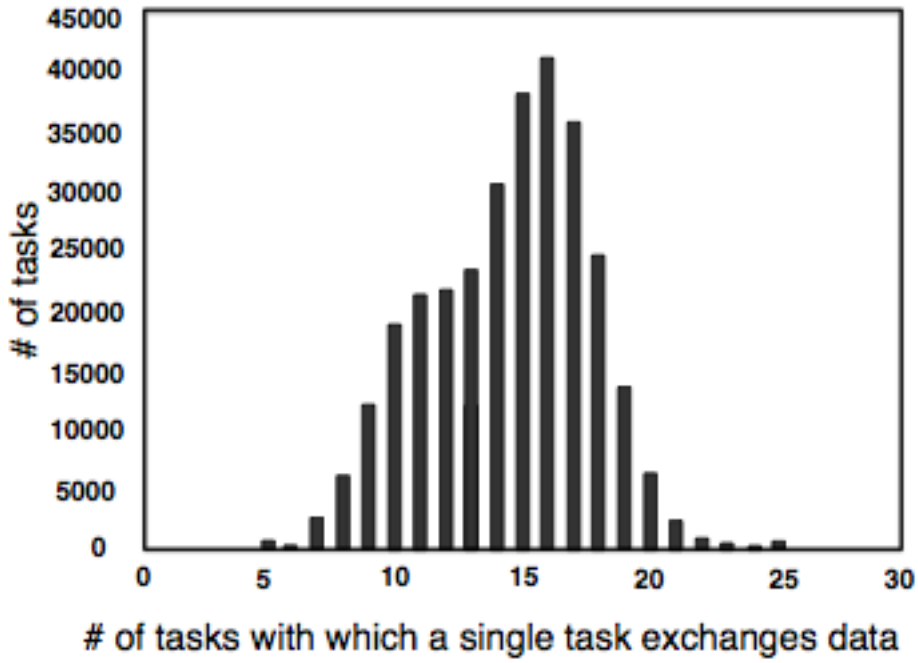


Figure 4.2: The distribution of the 294,912 tasks with respect to the number of tasks with which they are required to communicate

and highly varied morphology of the computational domains, often taking highly counterintuitive shapes in the vicinity of geometrical complexities. This *morphological richness*, which stands in stark contrast with elementary partitionings in idealized geometries (cubes, slabs and similar), conveys an intuitive flavor for the complexity of the partitioning task and also hints at some form of homeo-morphism between dynamics and geometry which surely deserves a separate investigation for the future.

We create the communication pattern by the following "run-time" pre-processing procedure. Mesh nodes are assigned to tasks according to the partition created as described above. Each mesh node is also labeled, in the input file, with a *tag* that identifies it as belonging to a specific subregion of the computational domain (*e.g.*, fluid, wall, inlet, outlet). After the assignment of the nodes to the tasks, the *pre-processing* phase begins. Basically, each task asks which tasks own the nodes to be accessed during the subsequent phases of simulation, for instance for the streaming part of the LB algorithm and for the Molecular Dynamics. Such information is exchanged by using MPI collective communication primitives, so that each task knows the neighboring peers for send/receive operations. Information about the size of data to be sent/received is exchanged as well.

All point-to-point communication operations make use of the same scheme: the *receive* operations are always posted in advance by using corresponding non-blocking MPI primitives, then the *send* operations are carried out using either blocking or non-blocking primitives, depending on the parallel platform in use (unfortunately, as

it is well known, few platforms allow real overlapping between communication and computation). Then, each task waits for the completion of its receive operations, using the MPI *wait* primitives. The latter operation, in the case of non-blocking *send* operations, is to wait for their completion. The choice between blocking and non-blocking *send* can be done at run time. The evaluation of global quantities (*e.g.,* the momentum along the x, y, z directions) is carried out by using MPI collective *reduction* primitives.

Molecular Dynamics with a highly irregular domain decomposition is a major challenge in itself. In most parallel Molecular Dynamics applications the geometry of the spatial domain is a regular bounding box with Cartesian decompositions defined in such a way that each task has (approximately) the same number of particles and minimal communicating regions. In our case, this strategy would generate two separate domain decompositions: one for the LB (defined by the graph-based partitioning method previously described) and another for the MD part of the simulation. As a consequence, the exchange of momentum between particles and fluid would become a non-local operation with a very high cost due to the long-range point-to-point communications imposed on the underlying hardware/software platform. For the IBM BlueGene such communications are explicitly discouraged. We decided to resort to a domain decomposition strategy where the MD parallel domains coincide with the decomposition of the LB mesh. In this way, each computational task performs both the LB and MD calculations and the interactions of the particles with the fluid are

quasi-local.

The underlying LB mesh serves the purpose of identifying particles that belong to the domain via a test of membership: a particle with position $\mathbf{R}$ belongs to the domain if the vector of nearest integers [round(R_x), round(R_y), round(R_z)] coincides with a mesh point of the domain. Additionally, the load balancing of the mesh partitioning into the MD component can be exploited, given that an even number of RBCs is expected to populate the domains. For the MD part of the code, a novel parallelization strategy suitable for the irregular geometry of the LB domains has been developed. The solution relies on the notion of cells, parallelepipeds with linear sizes greater or equal to the interaction cutoff, that cover the whole irregular domain. This representation allows the processors to *i)* perform an efficient search of both interdomain and intradomain pairs of particles and *ii)* to reduce data transfers by exchanging a limited superset of the particles actually involved in interdomain pairs and particles moving across domains.

The cells are grouped into three sets (*internal*, *frontier* and *external* cells) that verify the following properties:

1. Every point of the domain is within either an *internal* or a *frontier* cell;

2. *Internal* cells contain only points of the domain at distance greater than the cutoff distance from the domain boundary;

3. *Frontier* cells contain all the points of the domain at distance less than or equal

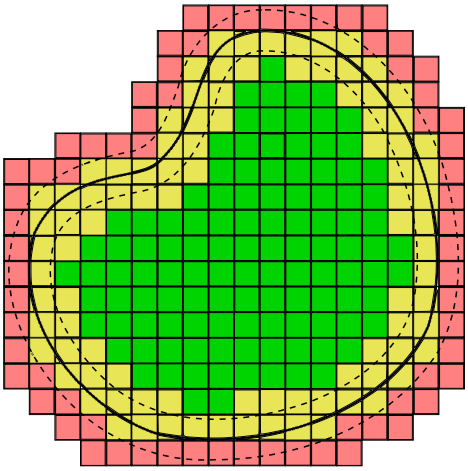


Figure 4.3: Decomposition of a 2D domain in *external* cells (red), *frontier* cells (yellow) and *internal* cells (green). The dashed line represents the region within a cutoff distance from the domain (solid line). The domain frontier has a staircase shape, but in this figure it is shown as a smooth curve for f simplicity.

to the cutoff distance from the domain boundary;

4. *External* cells contain only points outside the domain;

5. All external points at distance less than or equal to the cutoff distance from the domain boundary lie within either an *external* or a *frontier* cell.

Figure 4.3 shows an example of such decomposition applied to a simplified two dimension domain.

The decomposition into cells helps in handling MD for irregular domains in the following way. At the beginning of each iteration, each processor searches for the particles inside its domain that could interact with particles located inside neighboring domains. Property 3 guarantees that the particles are only contained inside *frontier* cells. All particles located in the *frontier* cells are exchanged with neighboring

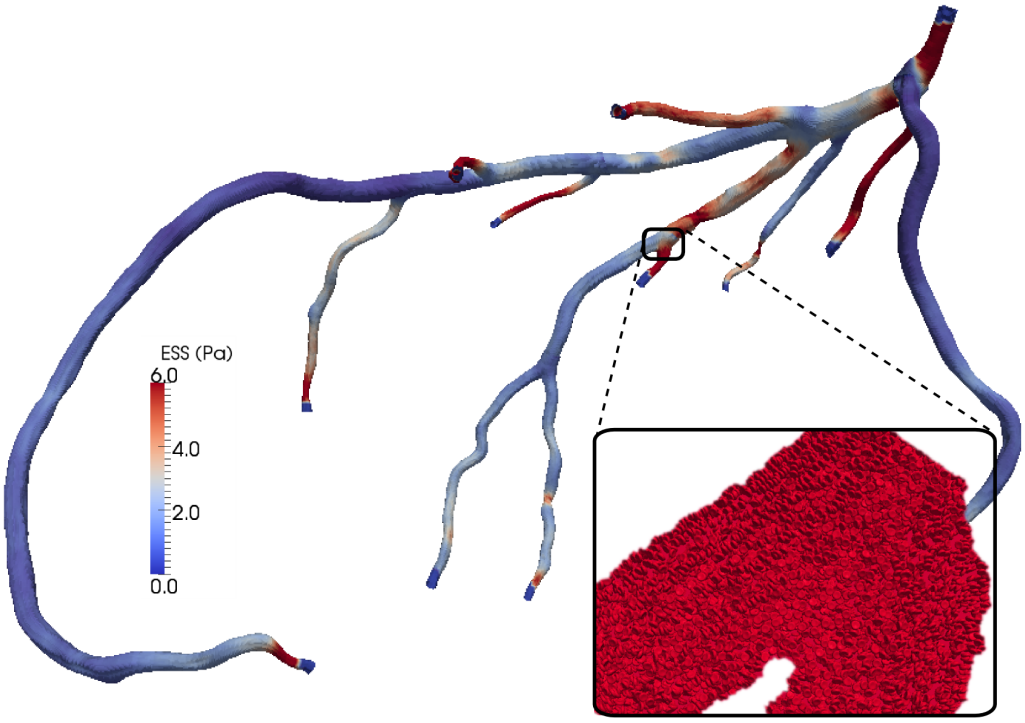


Figure 4.4: The geometry of the 12.5μm resolution test case, derived from a CTA scan of human coronary arteries. The inset shows a detail of the geometry with red blood cells visible. Note: the red color in the inset is meant simply to highlight the presence of RBCs and is not an indicator of ESS. The Endothelial Shear Stress (ESS) is the field derived from the simulations that encodes the atherosclerotic risk map and is represented as a color map on the arterial walls.

processors so that only a limited superset of the particles that could interact with the outer region is transferred. On the receiving side, only the particles that could interact with the inner region are considered. Given property 5, the received particles to be retained lie inside either *external* or *frontier* cells. After particles involved in interdomain pairs are exchanged, forces can be computed and particles positions updated.

Next, RBC migration among processors is handled. All particles are binned inside the cells they moved into, and those that left the domain are exchanged with neighboring processors. Departing particles are found by selecting those that moved to *external* cells (property 4) and *frontier* cells. To discriminate RBCs inside the frontier cells that moved to other domains, the underlying mesh is used by means of the membership test previously described. In this way, each processor sends exactly the particles that left its domain to all neighboring domains. On the receiving side, incoming particles are selected among the pool of all transferred ones.

The final component of the discussed multiscale methodology involves the fluid-particle coupling. Each suspended RBC experiences hydrodynamic forces and torques arising from the fluid macroscopic velocity and vorticity, smeared over a domain made of $4 \times 4 \times 4$ mesh points. This non-local operation requires a communication step such that each processor owning a given particle exchanges the hydrodynamic quantities with the surrounding processors. The same type of information is exchanged to build the forces acting on the fluid and arising from the suspended RBCs.

4.6 Results

The performance of the Lattice Boltzmann component of *MUPHY* on a single core is in line with other LB kernels highly tuned for the Bluegene/P platform[30]. From this viewpoint, it should be noted that: *i)* the algorithm for the update of the LB populations has an unfavorable ratio between number of floating point operations and number of memory accesses; *ii)* unlike other applications which can heavily draw upon consolidated computational kernels (*e.g.,* matrix operations or FFTs), no optimized libraries are available to perform the basic LB operations; and *iii)* it is not possibile to exploit the SIMD-like operations of the PowerPC 440 processor, since these require stride-one access whereas the LB method has a "scattered" data access pattern due to the streaming phase.

We focus on the total runtime for the simulation, as well as on the breakdown between computation and communication. To this end, a simulation was run at $12.5\,\mu$m resolution, corresponding to about 1 billion lattice sites for the fluid flow. All simulations were run over 200 time-steps. The measurements were performed on the Jülich Bluegene/P with $294,912$ cores, 144 TB of total memory and a theoretical peak performance of about 1 Petaflops. All runs were made in VN mode.

With a mesh having 1 billion fluid nodes within a bounding box having a total of almost 300 billion nodes, the reference hematocrit level corresponds to 10 million RBCs, run on the 72 racks system. More recently, results were obtained utilizing the same mesh but with 100 and 300 million RBCs on the 40 rack Bluegene/P system

at Argonne National Laboratory. These results provided a fundamental check of the reliability of the code at physiologic levels of hematocrit.

The successful completion of the simulations at each number of RBCs proves the feasibility and robustness of the method up to physiological levels. The joint usage of linkcell algorithms to compute pairwise interactions together with the linear method to access the indirect addresses of the mesh for the RBC-fluid exchange of hydrodynamic forces proved the linearity of the multiscale methodology with the problem size on the 40 racks installation. Through the combination of the fine mesh and inclusion of red blood cells, ESS in the patient could be observed over the course of several hundred time steps as shown in Figure 4.4.

4.6.1 Strong Scaling

This scaling analysis is performed by increasing the number of processors at a fixed problem size, in an effort to analyze the impact of the number of computational cores on the total simulation time. In Table 4.1 and Figure 4.5, we show the elapsed time per time-step, as well as the breakup for the LB and MD components separately. A few comments are in order. First, the elapsed time decreases significantly with the number of cores, with a speed-up of 43.5 between the $4,096$- versus $294,912$- core configurations (see Fig. 4.6), corresponding to a parallel efficiency in excess of 60 %. This result is particularly significant given that the average number of mesh points per computational core becomes pretty low (*i.e.,* $\sim 3,300$) on the full configuration

of 294,912 cores. Particularly efficient are the data concerning the LB component, showing a speed-up of 54.1 and efficiency of 75 %. Second, note that up to 147,456 cores the MD and LB sections remain in a fairly satisfactory balance with each other across the whole range of cores, thereby highlighting the excellent quality of the workload partitioning. Third, the MD component shows saturation above 147,456 cores. This is not unexpected, since at this number of cores and with 10 million RBCs each domain contains an average number of 60 RBCs, a critically small number regarding the calculations of Gay-Berne forces, the time-consuming MD component. Below the threshold of 147,456 cores, the constant ratio between the LB and MD workloads underscores the good response of the MD component in dealing with a handful of particles per domain. It is likely that, with a significant increase in the number of RBCs, the MD component would show a further speed-up up to 294,912 cores.

Table 4.1: Breakdown of the elapsed runtime

Cores	LB	MD	LB+MD
4,096	0.4761	0.04633	0.5224
16,384	0.1191	0.01610	0.1352
147,456	0.0151	0.00419	0.0193
294,912	0.0088	0.00419	0.0130

To further analyze the parallel performance of this simulation, the breakdown of

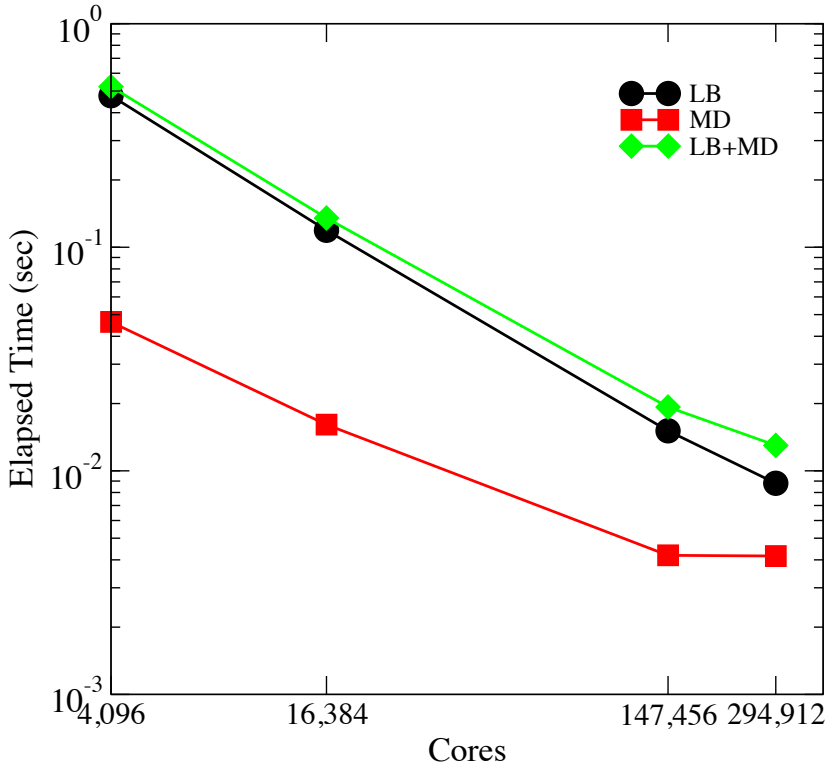


Figure 4.5: Log-log plot of the elapsed time for the LB component (circles), the MD component (squares) and for the full simulation (diamonds) versus the number of cores, for the system composed by 1 billion fluid nodes and 10 million RBCs.

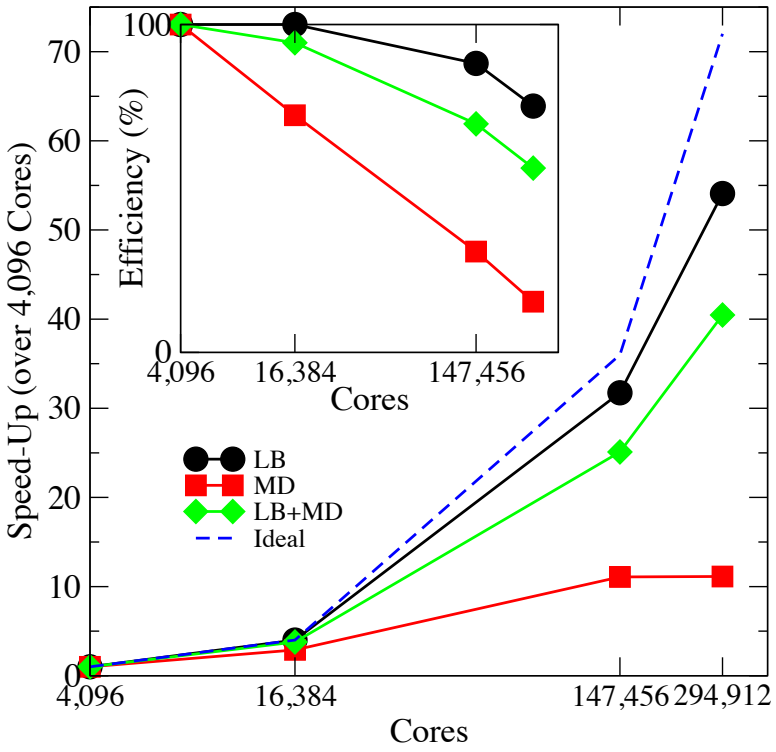


Figure 4.6: Semilog plot of the speed-up for the LB (circles) and MD (squares) components, for the full simulation (diamonds), and for the ideal regime (dashed line) versus the number of cores. Data are for the system of 1 billion fluid nodes and 10 million RBCs.

the communication time across the pool of cores was inspected.

Details of the communication performance were obtained using the MPI Profiler [82]. The communication times decrease significantly as the number of cores increases, roughly by 24% when going from $16,384$ to $294,912$ cores. The time for communication by the master core remains basically the same, with just a minor 5% increase. Table 4.2 reports the MPI function communication summary, where the Send/Irecv calls represent by and large the most time-consuming communication routines. The bandwidth for the (blocking) MPI_send corresponds to roughly 27 MB/sec and is satisfactory in view of the highly non-trivial communication pattern.

Table 4.2: Communication Breakdown for the run with $294,912$ cores.

MPI routine	Calls	Avg. bytes	Time(sec)
MPI_Send	10251	1187.7	0.452
MPI_Irecv	10302	17148.0	0.016
MPI_Waitall	603	0.0	0.222

4.6.2 Hardware Performance Monitoring

Finally, a thorough performance analysis was conducted using the hardware performance monitoring library (HPM) on BlueGene/P. HPM tracks 256 performance counters that measure events ranging from integer and floating-point operations to cache and memory accesses. The hardware counters can be set to measure the per-

formance for either cores 0 and 1 or cores 2 and 3 during a single execution. The tool reports Flops as a weighted sum of various floating-point operations. More information is given in [82]. For 72 racks of BlueGene/P in VN mode (294,912 cores), 64 TeraFlops were measured, as shown in Figure. 4.7. In view of the intrinsic Flop-limitations of the Lattice Boltzmann algorithm discussed previously, and taking into account the coupling between LB and MD components, this appears to be a fairly satisfactory overall performance. Just to convey the flavor of the practical impact of this application, the above performance corresponds to simulating a full heartbeat at microsecond resolution in only a few hours time on the 72-rack BlueGene/P system.

4.7 Discussion

Summarizing, presented here is the first large-scale simulation ever of the entire heart-circulation cardiovascular system, with a realistic representation of the complex human arterial geometry at the spatial resolution of red-blood cells: from centimeters all the way down to microns in a single multiscale simulation. This simulation, involving one-billion fluid nodes, embedded in a bounding space of three hundred billion voxels and coupled with the concurrent motion of ten million red-blood cells, achieves over 60 Teraflops performance on the full 294,912 BlueGene/P processor configuration, with a parallel efficiency in excess of 60 percent, performing about 100 billion lattice updates per seconds. Using the same arterial system and simulation parameters it has been possible to elevate the hematocrit level to physiological levels

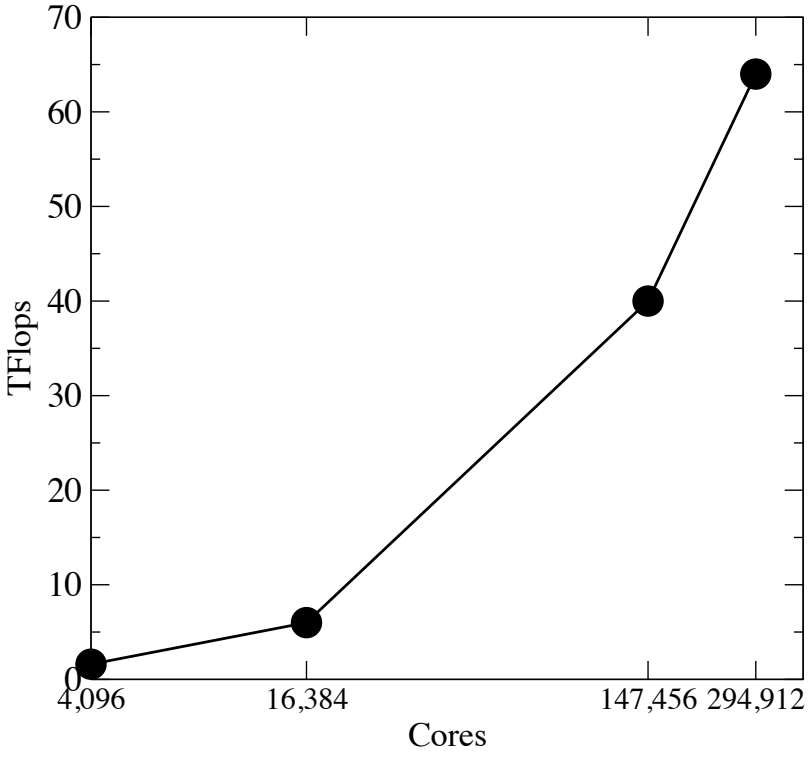


Figure 4.7: Aggregate performance (Floating Point Operations Per Second) as a function of the number of cores.

of 300 million RBCs.

The above achievement results from the development of several unique features, in terms of both high-performance computing technology and of physical/computational modeling, namely i) the solution of the formidable graph-partitioning problem prompted by the need of evenly distributing the workload associated with the complex arterial geometry, across as many as $294,912$ BlueGene/P cores; ii) the innovative communication techniques required to secure a balanced workload between fluid-dynamics and Molecular Dynamics in geometries of real-life complexity; and iii) the innovative modeling techniques required to manage the self-consistent fluid-particle interactions in complex geometries. As to (ii), we are not aware of any previous implementation dealing with non-ideal geometries.

This work represents major progress in the predictive capabilities of computer simulation for *real-life* cardiovascular applications. The scientific and societal impact of the extensions of such activity cannot be underestimated.

Currently, no combination of computational models, however sophisticated, can provide a comprehensive and all-embracing description of all complex phenomena which underlie the dynamical behavior of the entire human cardiovascular system, including a realistic description of the arterial tissues, wall compliance, RBCs deformability, to name but a few. However, this does not prevent the possibility to gain completely new insights on specific cardiovascular phenomena of major clinical relevance. For instance, as far as long-term atherogenesis is concerned, neither wall

compliance, nor RBCs deformability, are credited for playing a lead role. On the other hand, even in large arteries, the finite extent of the RBCs, is likely to exert a major effect on the near-wall circulation patterns, hence the local wall shear stress distribution. This is the kind of effect that the present simulations are expected to shed new light on, once appropriate hardware resources are available.

This research raises several questions that will be addressed in the following chapters. First, even taking into account all of the aforementioned strategies for efficient parallelization, the simulation of one heartbeat required the use of 163,840 cores for a full six hours. This order of time scale is not feasible to enable physicians to leverage simulations such as this on a regular basis. Furthermore, the inset of Fig. 4.4 shows the density of red blood cells for physiological values of hematocrit. This level of density causes the distance between the cells that the plasma moves between to diminish and cause the fluid model to approach the continuum limit. To address both of these concerns, focus was shifted to focus purely on the fluid component of the model. In the follow chapters, methods to optimize the lattice Boltzmann model for the D3Q19 velocity model presented here as well as higher order models that extend the accuracy of the simulation beyond the continuum limit are presented.

5

Fluid Models Beyond Navier-Stokes

> *But not finding it possible that this could be supplied by the juices of the ingested aliment without the veins on the one hand becoming drained, and the arteries on the other getting ruptured through the excessive charge of blood, unless the blood somehow flowed back again from the arteries into the veins and returned to the right ventricle of the heart. In consequence, I began privately to consider if it had a movement, as it were a circle.*
>
> – William Harvey, *De motu cordis [68]*

5.1 Motivation

The realization that blood actually circulated through the body was a great breakthrough in the history of the study of hemodynamics and as such the massively parallel application presented in this chapter is named HARVEY after its discoverer, William Harvey. In Chapter 4, a model of blood flow coupled with the motion of red blood

cells was introduced and the methods to provide efficient scaling up to 294,912 cores presented. As discussed, the model presented some challenges such as the length of time required to simulate just on heartbeat. To address such difficulties, the following chapter focuses on fine-tuning the lattice Boltzmann model to ensure minimization of the overall time to solution for the fluid model as well as ways to extend the regime for which it is accurate through the introduction of HARVEY.

Beyond the cardiovascular models, the increasing demand for micron scale simulations for devices such as those used for microfluidics, furthers the urgency of the need for models that can accurately model fluid flow beyond the continuum regime, and for the development of optimization techniques that will enable these models to achieve strong performance on current and future computer architectures. The objective of this work is to study the performance impact of improving the accuracy of a computational fluid dynamics (CFD) model and to identify methods to mitigate this cost, thus making the simulation of extreme regimes of CFD tractable.

As shown, a multiscale fluid dynamics simulation was developed that allows for the modeling of flow in complicated geometries from microfluidic devices to patient-specific arterial geometries obtained from computed tomography (CT) scans [113], [114]. Initial models have focused on flow in the coronary arteries where the diameters are on the order of millimeters as shown in Fig. 5.1. In this chapter, the model is extended such that it can be applied to other domains in fluid dynamics such as the study of clogging in a microfluidic device. In expanding the use of this application

to modeling gaseous flows in such devices, the model must be extended to accurately simulate flows beyond the continuum regime. This will enable us to study situations in which the traditional model may also fail for liquids, as in the case of modeling the plasma flow between the particulates in blood.

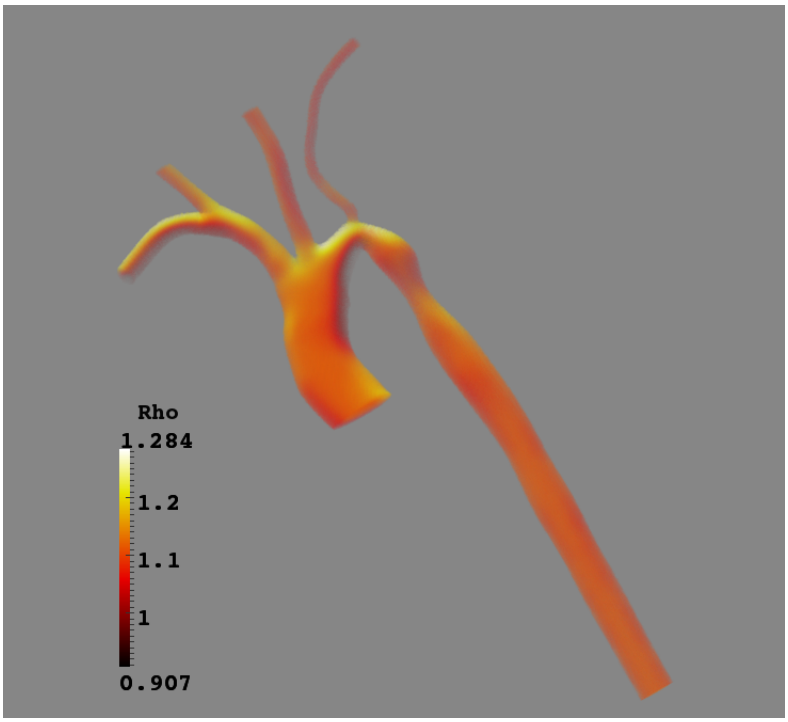


Figure 5.1: Fluid density in the aorta.

Traditionally, CFD methods for studying flow are based on the Euler or Navier-Stokes equations. These equations assume that the fluid is being modeled as a continuum; however, at small scales this assumption begins to break down and conventional CFD approaches become inaccurate [43]. The limit to the regimes accurately captured by these models are flows with Knudsen numbers (Kn) between 0 and 0.1 [26];

where $Kn = \frac{\lambda}{L}$ with λ being the average distance traveled by a molecule between collisions (the mean free path), and L the macroscopic length scale within which flow occurs. Beyond this range, many of the continuum assumptions break down and corrections are necessary as the contributions from higher kinetic moments are no longer negligible [171].

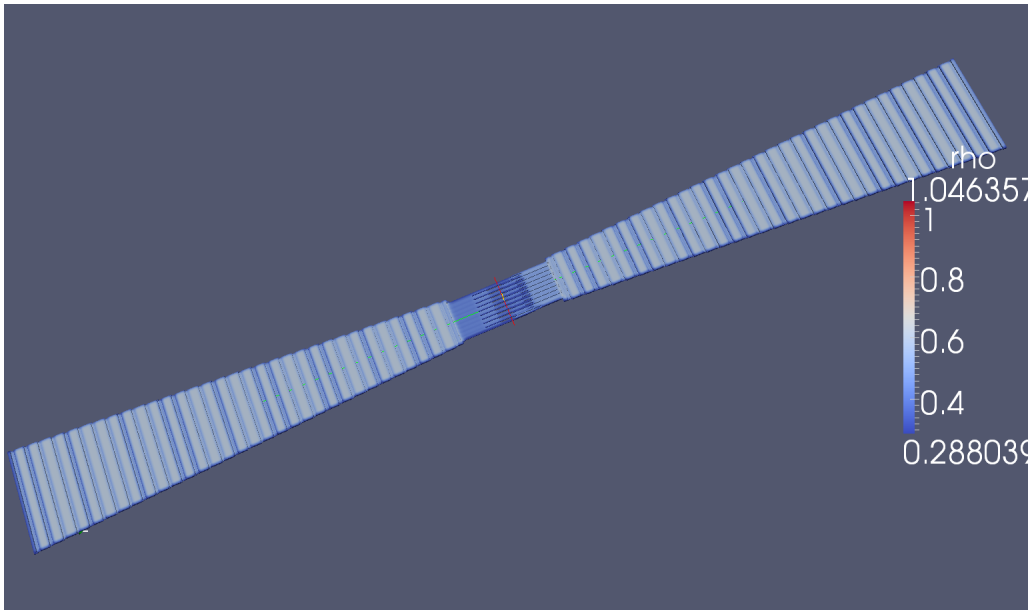


Figure 5.2: Microfluidic device.

Experiments have shown that the conventional methods may not produce accurate results for rarefied flows [3], [31], [4]. Alternative methods such as the direct simulation Monte Carlo [43], extensions to Navier-Stokes [140], [4], and use of the Burnett equations [169], have been investigated to address these situations. Due to its kinetic nature, the lattice Boltzmann method (LBM) offers a promising alternative

for simulating flows in which Kn falls outside the [0-0.1] interval. In this chapter, the focus is on on recent advances to the lattice Boltzmann model (LBM) that extend its reach to accurately model flows for larger Kn ranges such as those described by Chan, Yuan and Chen [139]. These higher order methods impact both the communication bandwidth and computational complexity of the application. The goal of this paper is to assess the impact of the higher order models on computational performance and introduce ways to mitigate this cost. The metric of success is defined as minimal wall clock time in seconds and maximal work units completed per second to enable the modeling of larger fluid systems in shorter physical time.

The hypothesis is that the use of deep halo ghost cells alongside further enhancements such as optimized data handling and structures, loop reordering and separation, branch minimization, and communication tuning will enable us to significantly improve the code's exhibited performance. First experiments are conducted to measure the effect of the code quality and impacts of singe node optimizations. Second, the impact that the message aggregation has on the communication performance is evaluated. Finally, experiments that address the challenges associated with scaling such as communication performance and threading with MPI/OpenMP are presented. Having defined the optimization levels, the methodology is validated through tests of each level on both the IBM Blue Gene/P and IBM Blue Gene/Q architectures. This analysis not only provides an upper bound of the potential performance metrics for targeted supercomputing architectures, but also highlights the increasing performance

restriction on the LBM due to the growing disparity between increases in bandwidth and flop rate on new architectures.

Significant performance results are presented: 83% of the predicted upper bound for Blue Gene/P and 79% on Blue Gene/Q. This correlated with a three-fold improvement on Blue Gene/P and almost an eight-fold improvement on Blue Gene/Q due to the optimizations for the extended models. Furthermore, it is demonstrated that models of extreme fluid flows through the extended LBM can be efficiently simulated on large-scale supercomputing platforms and that the computational and memory burdens can be mitigated through careful tuning alongside the use of special features such as deep ghost cells and hybrid programming models.

5.2 Adaptations to the Lattice Boltzmann Method

As mentioned, the fundamental quantity of the LBM is the particle distribution function, $f(x,t)$, that describes the likelihood of finding a fictitious fluid particle at lattice point x, at time step t, moving at the discrete velocity c_i. The particles move only along discretized velocity paths defined by the lattice. The distribution is evolved according to Eq. (5.1) [144]:

$$f(x + c_i\Delta t, t + \Delta t) = f(x,t) - \omega\Delta t(f(x,t) - f^{eq}(x,t)) \tag{5.1}$$

There are two key components to the algorithm: collision and advection. The collision step is calculated through a relaxation towards local equilibrium, as shown

in the right hand side of Eq. (5.1). In this work, we use the most common collision operator, the Bhatnagar-Gross-Krook (BGK), which relaxes to equilibrium on a single time scale [17]. The local equilibrium is defined as a truncated Hermite expansion of a local Maxwellian with density ρ and speed u [171]. The Navier-Stokes equation is recovered with a second order expansion:

$$f_i^{eq} = \omega_i \rho \left\{ 1 + \frac{\xi_i \cdot u}{c_s^2} + \frac{1}{2} \left(\frac{(\xi_i \cdot u)^2}{(c_s^2)^2} - \frac{u^2}{c_s} \right) \right\} \tag{5.2}$$

Higher-order expansions enable the physical effects beyond the continuum regime to be modeled. The third order accurate expansion is defined as the D3Q39 discrete velocity model, given by:

$$\begin{aligned} f_i^{eq} &= \omega_i \rho \left\{ 1 + \frac{\xi_i \cdot u}{c_s^2} + \left(\frac{(\xi_i \cdot u)^2}{2(c_s^2)^2} - \frac{u^2}{c_s} \right) \right. \\ &\quad \left. + \frac{\xi_i \cdot u}{6c_s^2} \left(\frac{(\xi_i \cdot u)^2}{c_s^2} - 3\frac{u^2}{c_s} \right) \right\} \end{aligned} \tag{5.3}$$

where ω defines the quadrature weight and c_s the speed of sound [146]. The added term in Eq. (5.3) is related to the velocity-dependent viscosity of the fluid. As discussed in [139], third-order truncation requires a discrete velocity model of sixth order isotropy as opposed to the fourth order needed for Eq. (5.2).

In this work, we focus on two velocity models. For continuum flow, we use the common 19-speed cubic D3Q19 lattice connecting each lattice point to its first and second neighbors [113]. The associated weights and discretized velocities are given in

Table 5.1: Parameters for the D3Q19 velocity model.

D3Q19 Lattice				
c_s^2	ξ_i	ω_i	Neighbor Order	Distance
$1/3$	$(0,0,0)$	$1/3$	0	0
$1/3$	$(\pm 1,0,0)$	$1/18$	1	1
$1/3$	$(\pm 1,\pm 1,0)$	$1/36$	2	$\sqrt{2}$

Table 5.1. To study further regimes, we employ a model using the next-order kinetic moments, the 39-point Gauss-Hermite quadrature defined in [139]. The associated weights and discretized velocities are given in Table 5.2.

The advection, or streaming step, involves propagating the fluid particles along the appropriate velocity trajectories. For the D3Q39 model, as opposed to the D3Q19 that focuses on up to second neighbors, particles can travel to lattice nodes that are as far away as the fifth nearest neighbor [146]. The velocities describe the 18 first and second neighbors or 38 first, second, third, fourth, and fifth nearest neighbors and the 19th and 39th values are for the lattice point itself, these are represented in the first row of the tables.

Table 5.2: Parameters for the D3Q39 velocity model.

D3Q39 Lattice				
c_s^2	ξ_i	ω_i	Neighbor Order	Distance
2/3	$(0,0,0)$	1/12	0	0
2/3	$(\pm1,0,0)$	1/12	1	1
2/3	$(\pm1,\pm1,\pm1)$	1/27	2	$\sqrt{3}$
2/3	$(\pm2,0,0)$	2/135	3	2
2/3	$(\pm2,\pm2,0)$	1/142	4	$2\sqrt{2}$
2/3	$(\pm3,0,0)$	1/1620	5	3

5.3 Systems

5.3.1 Platform Overview

The two platforms used in this paper are the IBM Blue Gene/P and IBM Blue Gene/Q architectures. Both rely on a system-on-a-chip backbone. The Blue Gene/P has a 32-bit PowerPC 450 processor that runs at 850 MHz. Each node consists of 4 cores capable of executing SIMD instructions when data is 16-byte aligned resulting in a peak performance of 13.6 GFlop/s. There are 2 GB of memory per node and 1 thread per processor, allowing up to four threads per node. Point-to-point communication

between nodes is handled via a 3D torus with a hardware (software) bandwidth per unidirectional link of 425 (375) MB/s [74].

Blue Gene/Q has a similar modular design but expands the options for threading, memory access, and speeds. It has a 64-bit PowerPC processor at 1.6 GHz. Each node consists of 16 cores with 4 potential threads per core. There is a 204.8 GFlop/s peak performance per node [67]. Memory per node is expanded to 16 GB and there is support for speculative execution and hardware assist to *sleep* threads while waiting for an event [25].

5.3.2 MFlup/s: A Performance Metric for the LBM

In order to determine the methods of optimization, it is first important to assess the bounds on performance expectations of our model for the platforms of focus.

When analyzing lattice Boltzmann performance, focusing on the flop/s is not the best metric as this can vary widely based on factors such as the implementation of the model, compilers, and hardware used. A more meaningful metric is the work done per unit time. For LBM, this means the number of lattice points updated per second. A standard measure for this is to measure *MFlup/s*, or million lattice point updates per second, which assesses the runtime of a production application depending only on domain size and number of time steps simulated. Equation 5.4 shows how the peak number of potential MFlup/s is calculated for a specific simulation. In this case, $T(s)$

refers to the execution time for s steps and N_{fl} defines the number of fluid cells [162].

$$P[MFlup/s] = \frac{s \cdot N_{fl}}{T(s) \cdot 10^6} \tag{5.4}$$

Based on the specific hardware details of each platform, we can calculate the maximum performance attainable of Eq. (5.4) and determine the performance limiting factors for our model in terms of bandwidth vs. computation. The application performance will either be limited by available memory bandwidth or peak performance. To calculate the attainable maximum performance P in MFlup/s, we use Wellein et al.'s model defined with Eq. (5.5) [162], in which B is the number of bytes per cell transferred to and from main memory and F is the number of floating point operations per cell.

$$P = \frac{Bm}{B} || \frac{Ppeak.}{F} \tag{5.5}$$

In this implementation there are two load operations and one store operation for every velocity mode. For the D3Q19 model, this results in $B = (19+19+19)*8 = 456$ bytes per lattice point while for the D3Q39 model, there are 936 bytes per lattice point. For the calculation of P in Table 5.3, we use the main store bandwidth measurement for individual compute nodes to obtain the maximum attainable performance. Inherently, we will see production results below these values as those implementations span multiple nodes and require point-to-point communication over the torus.

Both systems are capable of performing a maximum of four double precision floating-point operations (two multiply and two add) per cycle. To get the high baseline, we assume maximal use of this functionality in this performance model.

This is clearly an overstatement as the stream function consists primarily of load and store operations while the collide function has a high number of addition operations. For the D3Q19 model, our implementation has 178 core floating-point operations and for the D3Q39 model, it has 190 core floating-point operations. These rates do not depend on problem size.

The subsequent estimates for maximum achievable performance on these two platforms are shown in Table 5.3, in terms of peak MFlup/s given the main store bandwidth of the system and the peak given the flop/s for each processor. The calculated max MFlup/s are shown with the limiting factor for each system highlighted in red. Similar to the previous studied architectures, the bandwidth imposes the performance limit on each system.

5.3.3 Impact of Bandwidth Limitations

The fact that both models for both architectures are bandwidth limited indicates that the stream function is the limiting function as it consists of the bulk of the load/store operations in the movement of the particles. While the overall runtime can be reduced through arithmetic optimization of the collide function, scalability will be inherently limited by the memory bandwidth and therefore by the stream function. When extrapolating beyond the single node performance, the data is retrieved via point-to-point communication on the torus. Assuming all loads and stores occur at the torus bandwidth provides a lower bound for parallel performance. For D3Q19

Table 5.3: Table of the maximum MFlup/s attainable on the IBM Blue Gene/P and IBM Blue Gene/Q systems for both lattices with performance limiters highlighted in red. In all cases, the code is extremely bandwidth limited. The hardware system data for the IBM Blue Gene systems comes from [74], [67], and [25].

D3Q19 Lattice				
System	**Bm**	**P(Bm)**	**Ppeak**	**P(Ppeak)**
BG/P	13.6 GB/s	29 MFlup/s	13.6 GFlop/s	76.4 MFlup/s
BG/Q	43 GB/s	94 MFlup/s	204.8 GFlop/s	1150 MFlup/s
D3Q39 Lattice				
System	**Bm**	**P(Bm)**	**Ppeak**	**P(Ppeak)**
BG/P	13.6 GB/s	14.5 MFlup/s	13.6 GFlop/s	71.5 MFlup/s
BG/Q	43 GB/s	45 MFlup/s	204.8 GFlop/s	1077 MFlup/s

this falls at 11.1 MFlup/s and 70 MFlup/s for BG/P and BG/Q respectively. As for D3Q39, the lower bounds are at 5.4 MFlup/s and 34 MFlup/s. Of course this is an overestimate and a real code will have a mix of various accesses to various cache levels as well as communication, but this provides a crude view of performance expectations that is surprisingly useful.

This goal of this analysis is simply to provide insight into the limits of potential performance tuning and therefore give greater context to the results of the previously discussed optimizations. The P(Fp) defines the number of MFlup/s that would be attained at the peak flop rate. The ratio of P(Bm) to P(Fp) provides the upper bound on potential hardware efficiency. For Blue Gene/P, the models have the potential of achieving 38% (D3Q19) and 20% (D3Q39) hardware efficiency. While some applications achieve greater than 60% efficiency, most production parallel codes only leverage at most 10% of the available flop/s. This makes LBM ripe for high efficiency on such platforms. It is worth noting that the off node memory accesses have a less steep drop off in bandwidth, so as the code is highly parallelized, there will be less of a performance impact.

This model is, of course, over simplified and contains many assumptions, however, it does provide strong ground for assessing the upper bound of potential performance on new architectures and targeting optimization efforts. In our case, it is worth noting that the ghost cell implementation will add computation cycles not accounted for in the flop/flup ratio.

5.4 Implementation

In the work presented here, the goal is to assess the direct impacts on the computational performance due to algorithmic changes necessary to simulate fluid flow at finite Kn. To this end, all simulations in this work are of a cubic fluid system with periodic boundary conditions. This assumption allows the analysis to focus on the impact of the higher order terms and extended neighbors of the lattice instead of being dominated by boundary conditions. We further limit the study to a three-dimensional fluid system with one-dimensional domain decomposition. While this could restrict performance at the large-scaling limit, it again shifts focus to the algorithm and more specifically enables direct analysis of ghost cell depth impact. As the function of this code is to serve as the fluid component in a multiphysics coupling of red blood cell and blood plasma motion, the realistic domain decomposition will be irregular and rely on neighbor lists. Furthermore, it's been shown that cubic blocking can lead to overhead for long thin channels such as the geometries we would expect in artery and capillary blood flow models [162]. The code discussed in this paper is written in C and uses MPI and OpenMP for the parallelization.

As mentioned earlier, an LBM simulation consists of alternating steps completing the streaming and colliding of particle populations. A straightforward implementation of the method is shown in Fig. 5.3, in which a starting distribution of fluid particles is initialized, the stream function propagates the particles to adjacent lattice points and store these values in a temporary distribution function distr_adv. The collide

```
read initial distr
for (n< max_steps) {
  distr_adv=stream(distr);
  LBM_Exchange();
  distr = collide(distr_adv);
}
```

:

Figure 5.3: Naive implementation of the LBM.

function subsequently reads distr_adv, determines all resulting collisions, relaxes the population towards equilibrium, and updates the original array. In this way, it acts as a general stencil code using information from it's neighbors to update it's value and then pushing it's data to the neighbors, however, the data accessed in distr_adv is from another phase space.

Rüde, Pohl, and Wellein have extensively studied optimal data structures and cache blocking strategies for the BGK model (c.f. [162], [121], [122]). In this implementation, the *collision optimized* layout that they describe as optimal is leveraged.

In order to maximize messaging performance and set the code up for an easy transition to the use of indirect addressing necessary for irregular domains, the distribution functions were stored in two dimensional arrays of $(NumVelocities, zDim \cdot yDim \cdot xDim)$ allocated in contiguous memory. [113].

```
for ix < xDim
for iy < yDim
for iz < zDim {
  for is < numVel {
    ixa=ix+icx[is]
    iya=iy+icy[is]
    iza=iz+icz[is]
    boundary_counditions();
    distr_adv[is][iza+iya*Lz+ixa*Lz*Ly]
        = distr[is][iz+iy*Lz+ix*Lz*Ly]
  }
}
```

:

Figure 5.4: Stream pseudocode. The icx, icy, and icz arrays define the velocity directions, ξ_i.

Fig. 5.4 shows the details of the stream function. For each lattice point, all potential velocities are iterated over. The component of the velocity, ξ_i, is added to the correlating component of the lattice point coordinates. For example, particles with velocity $(1, 0, 0)$ at lattice point $(0, 0, 0)$ would stream to position $(1, 0, 0)$. The resulting distribution of the streaming step is stored in the temporary data structure of distr_adv.

The collide step is outlined in Fig. 5.5. For each lattice point and for each discrete

```
for ix < xDim
for iy < yDim
for iz < zDim {
  for is < numVel {
    calc_rho_and_vel()
    BGK=calc_BGK_op
    distr=update(distr_adv,BGK)
  }
}
```

:

Figure 5.5: Collide pseudocode.

velocity, macroscopic quantities of ρ and u are calculated locally. These values are then used to determine the relaxation towards equilibrium via the aforementioned BGK collision operator. Finally, the distribution array is updated. Note that the collision step relies on information from the neighboring processes stream function due to the fact that the stream function can result in particles displacing to lattice points contained on neighboring processors.

5.5 Optimizations

A precise simulation of fluid flow using either velocity model is demanding and requires well-optimized and scalable code. In this section, the sequential and parallel optimizations employed are presented.

5.5.1 Deep Halo Ghost Cells

As the update in the collide function requires data from the distr_adv array from all neighboring processors, this can lead to a communication bottleneck. A commonly employed tool to alleviate this contention at the boundaries is to add *ghost cells* or *halo cells*. The addition of this ghost layer increases the distribution array size by one in each direction of domain decomposition. At each time step, the neighboring processors exchange a copy of their border cells and receive the borders falling in their own ghost cell regions as shown in Fig. 5.6. Each processor adds an extra row to its domain of interest, as shown in blue, and populates this ghost cell row with the border data from its neighboring processor.

The use of an extra row of ghost cells is often found in large-scale models [165], [162]; however, by stopping at one row, potential for further tuning is being left unexplored. Kjolstad and Snir discussed implementation methods of the ghost cell pattern in [76] and suggested the investigation of *deep halos* as a potential method to trade off computation for communication. A deep halo refers to the use of ghost cell depth greater than one. Deep halos can be leveraged to further offset message

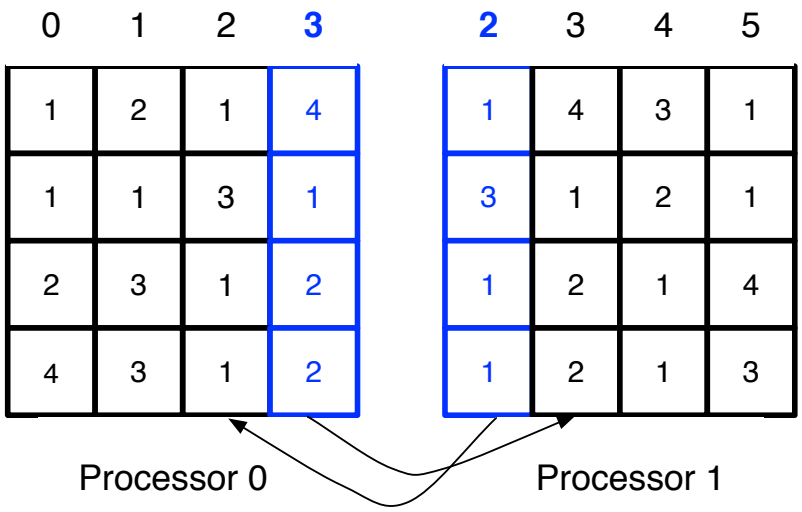


Figure 5.6: 2D example of ghost cells in x-dimension. Each processors receives a row from its neighboring processor to be used in the stencil calculation.

latency by reducing the number of overall messages used in a simulation. While this requires extra computation to update the ghost cells, in some cases the benefit from message reduction and further overlap of communication and computation can make this advantageous. By increasing the number of ghost cell width by a factor n, the data exchange can be minimized to only be required every n steps [76].

Note that for the D3Q39 lattice model, a deep halo implementation must be used simply for correctness. As mentioned previously, this model allows particles to move to neighboring grid points that are further away within one time step. The fundamental ghost cell depth must be set to include the number of neighbors that a particle could move within a time step (k). Discussions of ghost cell depth for the D3Q39 refer to the multiples of k included. For example, a ghost cell depth of 2

would include $2k$ additional cells at each side of a border exchange.

In a later section, the role that deep halo exchanges have on the performance of both velocity models investigated in this paper is discussed.

5.5.2 Data Handling (DH)

A thorough set of standard optimizations regarding the handing of the data was employed to improve performance. One of the overarching goals was to reduce the number of floating point operations in the two most intensive routines: stream() and collide(). Temporary variables were introduced to remove any redundant computation and arithmetic division was replaced with the multiplication of the reciprocal due to the heavy cycle count associated with division operations.

In this case, the largest impact came from optimal cache usage. Loops were restructured to both maximize cache reuse and reduce any recalculation. As mentioned earlier, the discrete velocities of the distribution function, $f(i)(zdim \cdot ydim \cdot zdim)$ are located contiguously in memory. To maximize cache reuse, the loops were reorganized such that all velocities are iterated over followed by the z-,y- and x- coordinates in memory order.

This was a moderate impact on performance on the Blue Gene/P architecture, 30%, but a very significant impact of an 75% increase in MFlup/s on Blue Gene/Q. This is due to the extensive cache hierarchy. In the original implementation, almost no loads during the collide function hit in the L2 cache while 3% hit in DDR. After

the DH tuning, there was a .4% increase in L1 d-cache and L1P buffer hits and a 1.2% increase in L2 cache hits while DDR dropped to .01%. This resulted in a longer load latency in the original version. During the stream function, cache hits were now optimized to fall only in L1 d-cache and the L1P buffer. These measurements were taken with the IBM Hardware Performance Monitor [58].

5.5.3 Compiler Optimizations

The impact that various XL/C compiler optimizations had for this application compared to the default $O3$ optimization level were assessed and shown to provide better loop scheduling and memory usage. It was shown that the most aggressive optimization level of $O5$ produced the strongest results while maintaining correct results, surpassing that of $O3$. While compilation took longer, the benefit gained was worthwhile. The most improvement, however, was gained through optimization of the intra-procedural analysis (IPA). By setting the qipa level to 2, whole-program alias analysis was enabled including the disambiguation of pointer dereferences and indirect function calls and whole program data reorganization. This resulted in significant performance gain while accuracy was maintained.

For the BG/Q implementation, it was found that a lower optimization setting of $O3$ produced better results increased the produced MFlup/s by 2.5x. By investigating generated lst files, we found that the compilers were more successful in automating loop unrolling and optimizing the floating-point instructions for this architecture.

5.5.4 Loop Restructuring and Branching Reduction (LoBr)

To further minimize the over all runtime of the application, we restructured the loops and reduced any branching in the code. With the addition of ghost cells to the simulations, especially those of deep ghost cell levels, there are several distinct sections of the domain to be modeled on each processor. In the case of a 1D domain decomposition, there is the ghost cell region from the previous processor, the local domain of interest, and the ghost cell region covering data from the next processor. In the LBM, both the stream and collide functions must iterate through loops that cover all three regions. We found that by explicitly separating these into different for loop groupings, we were able to better take advantage of the cache and minimize index calculation.

More improvement was garnered through a branch reduction trick we developed to swap if statements with for loops. This is outlined in Fig. 5.7. We removed all if statements from the innermost loop and replaced them with a for-loop that is able to continue without stalling. The location of where a particle is displaced to determines which region the new index falls in. For storing data, there is an offset needed to store ghost cell data. We create an array of these new indices based on the x-index in the outermost loop. This array is then iterated over for 1, 2, or 3 passes depending on the number of regions being spanned.

```
for ix < xDim {
  ixa=ix+icx[is]
  count = 0;
  if (xmin<  ixa < xmax)
    index[count] = (ixa-my_xmin+GCS)*LyLz
    count++
  if (gc_min1<  ixa < gc_max1)
    index[count] =  (ixa-gc_min1)*LyLz
    count++
  if (gc_min2<  ixa < gc_max2)
    index[count] =   (my_Lx-GCS+ixa-gc_min2)*LyLz
    count++
  for iy < yDim {
    iya=iy+icy[is]
    for iz < zDim{
      iza=iz+icz[is]
      for is < numVel {
        boundary_counditions()
        a = iz+iy*Lz+(ix-my_xmin+GCS)*LyLz
        for (jj=0; jj < count; jj++)
          distr_adv[is][iza+iya*Lz+index[jj]] = distr[is][a]
} } } }
```

:

Figure 5.7: Stream pseudocode with branch optimization.

5.5.5 Nonblocking Communication

In the naive implementation, blocking communication was used to exchange data between processors. Instead, this was switched to the use of non-blocking MPI functions. Especially for the non-ghost cell case, there is no opportunity to allow an overlap of computation and communication as the collide function directly relies on the results of the stream function from it's neighbors. The MPI_Irecv is posted before the local stream calculation and the MPI_ISend posts at the completion of the local stream. This results in a small reduction in the communication overhead that will be shown in the Results section. In the ghost cell implementation, the data can be sent at the end of the time step and waited on before the next stream function commences.

5.5.6 Separate collide function for collide (GC-C)

When using ghost cells, and especially when using deeper halos, the computation/communication overload can actually be increased by a much further degree. A separate function to handle the collision phase of the ghost cell regions was introduced. As the data being sent to the processor's neighbor is the border region of the domain of interest on that processor, it can be calculated and sent before the ghost cell region collisions are computed. By separating out the handling of the ghost cells and the region of interest, the message latency is hidden by overlapping it with the ghost cell computation. This is outlined in Fig. 5.8.

```
for (n < maxsteps) {
  read initial distr
  for (i < num_velocities) {
    for (z <z_dim)
    for (y < y_dim)
    for (x < x_dim) {
      if (n%GCL == 0) {
        MPI_Send
        MPI_Waitall
      }
      distr_adv = stream()
      distr = collide(distr_adv)
      if ( (n+1) % GCL ==0)  {
        MPI_Irecv(distr);
      }
      distr = gc_collide(distr_adv);
} } }
```

:

Figure 5.8: Separate handing of ghost cell collision.

5.5.7 SIMD Vectorization

Examining the compiler generated code for both BG/P and BG/Q, showed that the compiler failed to have SIMD double hummer intrinsics leveraged, therefore cutting the potential hardware efficiency already in half. To maximize performance, the code was modified to explicitly generate double hummer intrinsics through direct calls to instructions like fpmadd. This required enforcing 16-byte alignment and the disjoint pragma. Alongside the intrinsics, XL/C pragmas were utilized to force loop unrolling of the innermost loops in the functions [58].

For Blue Gene/Q, several compiler options were attempted as well as hand coding the intrinsics functions. In this case, there were modifications to the compiler so this work needed to be re-implemented for BG/Q instead of BG/P. Also, BG/Q has the expanded ability to handle different data alignments than simply the 16-byte alignment required for BG/P. Specifically in the collide function, quad-word load, store, and arithmetic operations were able to be used. While more limited, instructions like fused multiply-add were able to be taken advantage of. Without moving to vector doubles, we were not able to fully exploit QPX instructions [58].

5.6 Results

To assess the impact of the various optimizations previously discussed, Fig. 5.9 shows the results of progressive tuning of the two velocity models on each hardware

and their approach to the peak performance rate defined by our performance model. Performance is presented in terms of the previously discussed quantity, MFlup/s. For Blue Gene/P, the MFlup/s achieved for D3Q19 is 92% of the peak performance from our model. The slight discrepancy can be partially accounted for in that the model is actually targeting single node performance while Fig. 5.9 depicts results from multi-node runs. This difference was intentional in order to enable side-by-side comparison of the single node optimizations with the communication improvements. It does, however, introduce the previously discussed performance degradation from use of the torus for communication instead of all on node memory access. The torus has a lower bandwidth and will reduce the achieved MFlup/s. Moreover, the optimal runtime was achieved using the ghost cell method, which adds lattice updates not accounted for in the performance model.

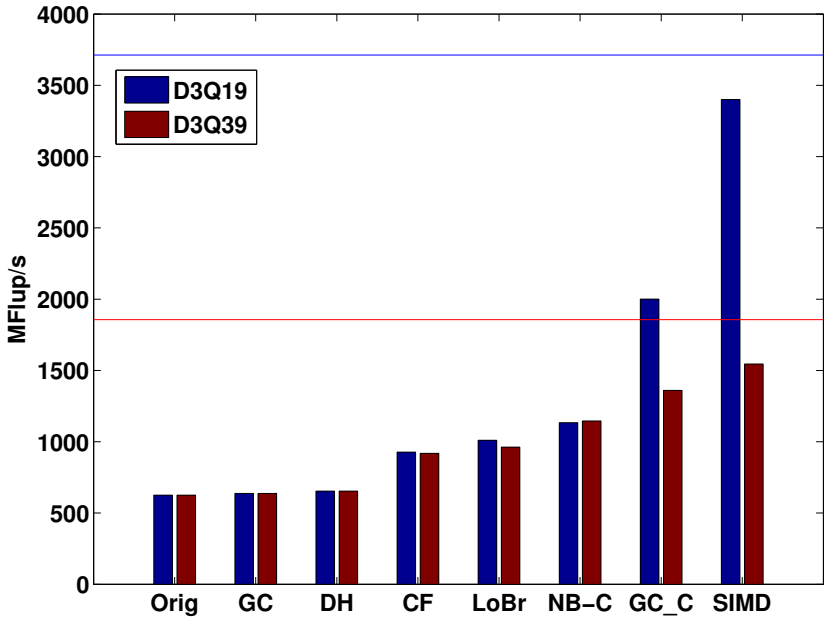
Additionally, the model shows a maximum hardware efficiency of 38% and with these optimizations, we achieve 31% of peak flop/s for the full simulation and 43% hardware efficiency in the compute heavy collide routine. This further confirms that the optimizations discussed have tuned the code almost to its maximum potential.

For D3Q39, it was slightly lower at 83% of the peak predicted performance value, likely due to the increased impact of the ghost cell implementation. In this case, 2 extra boundary rows are added around each processor boundary. The additional cost of these lattice updates are not accounted for and introduce a larger impact on the overall performance. The optimizations for this level with the largest impact were

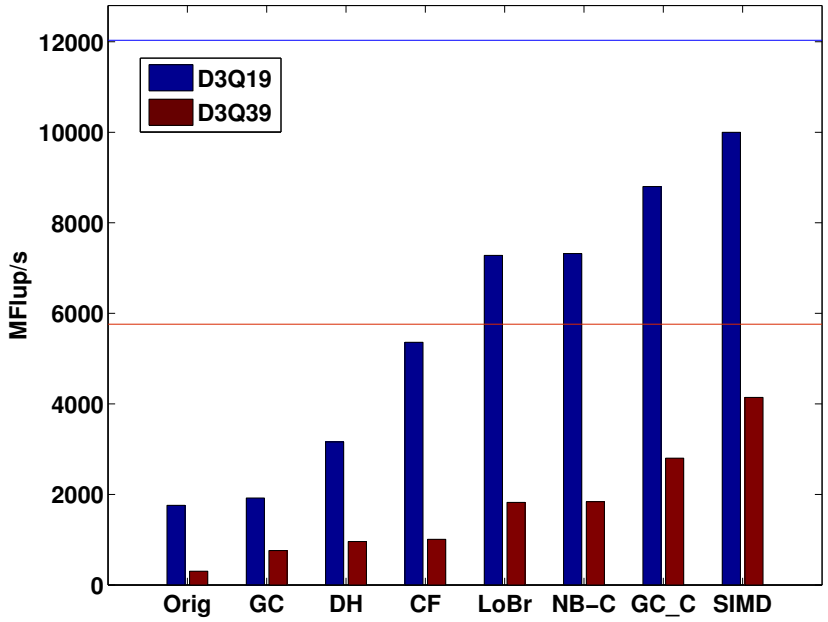
the compiler settings and the separate collide function for ghost cells. This is likely due to the extended number of ghost cells providing a more substantial option for communication/computation overlap. We will investigate impacts on communication overhead later in this section.

As for Blue Gene/Q, the largest impacts came from the compiler optimizations and the data handling. The intrinsics provided less of an impact, likely for two reasons. First, much of the performance gain was already achieved through the compiler and BG/Q gains a lot of its performance bump from the Quad Processing Extension (QPX) built-in functions. In the current implementation, we do not use the vector logical functions, leaving room for potentially further tuning. Optimal use of the cache and compiler optimizations proved the most fruitful optimizations. The max issue rate per core rose from 16.19% to 29.52%, meaning that each core is producing instructions at about 30% of the theoretical limit. This value is a good issue rate especially considering these results are from 128 nodes using 32 tasks per node with an unthreaded implementation. As shown in Fig. 5.9, the tuned version of the code approached the estimated performance maximum. For overall performance of the D3Q19 and D3Q39 models, we recovered 85% and 79% of the peak predicted performance. Again, these results are from a multi-node partition, introducing the degradation from intra-node communication.

Some of the optimizations encapsulated in Fig. 5.9 improved the load balance of the application and consequently the parallel nature of the application more than the



(a) Blue Gene/P Optimization Impacts.



(b) Blue Gene/Q Optimization Impacts.

Figure 5.9: MFlup/s achieved with each optimization enhancement on the two platforms in question. The horizontal lines represent the corresponding peak MFlup/s. In each case, 128 nodes were used.

performance as measured by MFlup/s of a single processor count. To gain insight into the impact of the communication tuning, we look at the time spent by the node spending the minimum, median, and maximum time in communication. This data, presented in Fig. 7.5, shows the communication balance of simply using non-blocking communication with solid lines. The blue lines refer to the D3Q19 model and the red to the D3Q39 model. The sharp slope of both lines indicates the strong load imbalance as one node spends as little as 4.8 seconds in communication while another spends 40 seconds almost entirely in MPI_Waitall. The dash-dot lines represent the use of both non-blocking communication and ghost cells. The introduction of the ghost cells allows the data to be sent at the end of the time step instead of causing the collide function to wait for the results of the stream function of neighboring processors. While there is still limited overlap with computation, communication imbalance is reduced. Finally, the dashed lines show the improvement gained through the introduction of a separate collide function to calculate the ghost cell data. This function allows the sends to be posted before the ghost cell calculations. As the receives can be posted at the beginning of the time step, the latency of the message passing can be hidden by the time for computing the ghost cells. The communication imbalance is minimized to ranging from 3-5 seconds for the D3Q19 model for example, posing a significant improvement to the initial range of 4.8-40 seconds. Simulations conducted for a greater number of time steps and larger fluid system sizes saw roughly the same ratio to hold throughout.

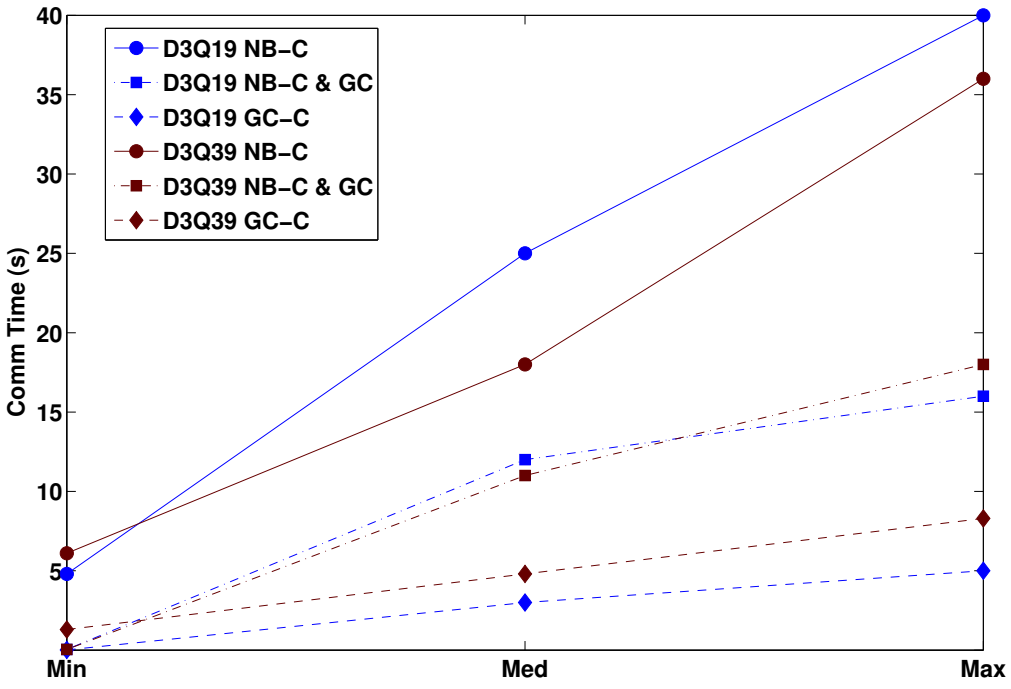
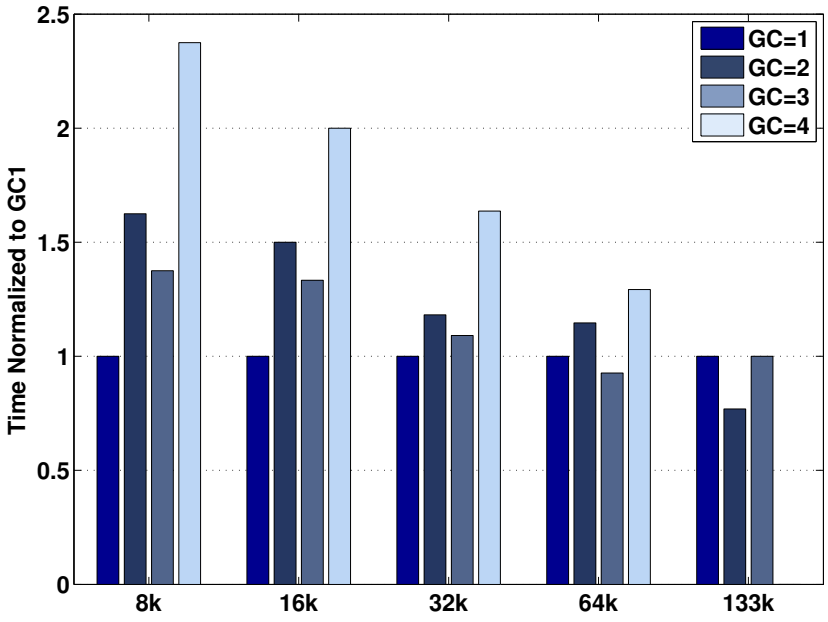


Figure 5.10: Time in seconds spent in communication for the processors that exhibited the minimum, median and maximum communication time at a range of optimization levels.

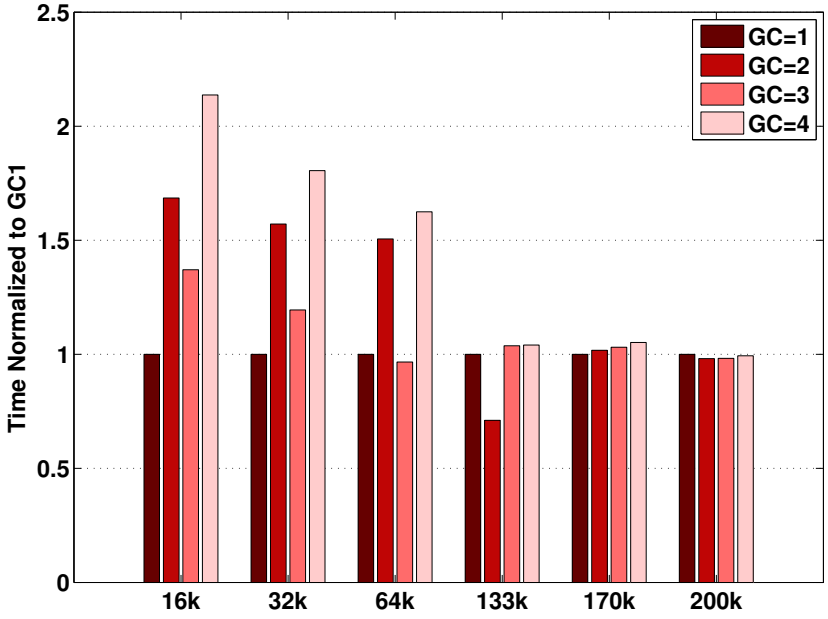
5.6.1 Deep Halo Ghost Cells

The use of deep halo ghost cells can further reduce message overhead by reducing the number of messages being sent. A greater number of ghost cells are retained on each processor, subsequently introducing a small computational cost, but messages are only exchanged every few time steps. The same amount of data is passed, but the reduction in number of messages allows for easier masking of the messaging latency. In order to assess the tradeoff between the communication gains and added computation, we simulated several different fluid system sizes for 300 time steps, enough so that the messaging tradeoff would have a visible impact on the runtime. For the D3Q19 model, 2048 processors on Blue Gene/P were used. The results given in Fig. 5.11 are normalized to the runtime for one ghost cell. GC refers to the ghost cell depth. Again, note that GC=1 for the D3Q39 model actually includes two extra lattice points in that direction as particles can move up to two points in a single time steps. The results highlight that at small population counts, the ghost cells have a higher impact on the surface/volume ratio and lead to typically longer runtimes. It is not until the larger sizes of 64,000 and 133,000 fluid nodes that a 2-ghost cell deep and 3 ghost cell deep implementation becomes optimal. The size indicates the size of the dimension being partitioned across processors. The other dimensions are held constant for the purpose of this study. For the 133,000 case, the individual nodes ran out of memory due to the addition of the fourth ghost cell and could not complete the simulation.

For the D3Q39 model, system sizes that fit into memory on BG/P were not large



(a) D3Q19



(b) D3Q39

Figure 5.11: Results showing optimal ghost cell depth, GC, at a variety of fluid system sizes. The results for the D3Q19 model were obtained on 2048 processors of Blue Gene/P while the results for the D3Q39 were from 16 nodes on Blue Gene/Q run with 16 tasks and 1 thread per node. This difference was due to differences in memory constraints between the two models.

enough to overcome the added cost of computing an additional 2 lattice points in the direction of each neighbor for each ghost cell level, so 16 nodes on Blue Gene/Q were used with 16 tasks each and one thread. Fig. 5.11 shows the results for dimensions ranging from 16,000 to 200,000.

In both graphs of Fig. 5.11, deep levels of ghost cells are shown to be beneficial at various fluid sizes and can produce more efficient simulations. For example, with the D3Q19 model ghost cell=2 for 64k corresponds to a hardware efficiency of 27% and 43% efficient in the collide routine, achieving several percent higher than seen with other fluid sizes.

For both Blue Gene/P and Blue Gene/Q, the number of ghost cells ideal for the D3Q19 model depends on the ratio of the dimension of the fluid system to the number of processors. The ideal ratios in Table 5.4 were consistent for both architectures, however, ratios beyond 66 per node were unable to be tested on either due to memory constraints. For systems with larger memory footprints, further lower bounds that require more ghost cells would likely be identified.

On Blue Gene/P, the memory overhead associated with deep halo ghost cells of the D3Q39 model made it have no performance gain. On Blue Gene/Q, however, the deeper levels started to have an impact mimicking the results shown for the D3Q19 model. Again, the most efficient level did not simply increase linearly with the ratio as one might naively expect. Due to on-node memory restrictions, ratios beyond 800:1 were not able to be tested. At higher ratios, it is likely that even deeper ghost

Table 5.4: Optimal ghost cell depth for fluid size/processor ratios in the D3Q19 lattice model.

Lattice Points/Proc	**Ghost Cell Depth**
$R \leq 16$	1
$16 < R \leq 32$	3
$32 < R \leq 66$	2

cell depths will be beneficial. At the maximum ratio tested here, the impact of 2 vs. 3 ghost cell layers was negligible.

Table 5.5: Optimal ghost cell depth for fluid size/processor ratios in the D3Q39 lattice model.

Lattice Points/Proc	**Ghost Cell Depth**
$R < 256$	1
$532 < R \leq 256$	3
$680 < R \leq 532$	2
$800 < R \leq 680$	2 or 3

5.6.2 Hybrid Implementation

We finally studied the role that a hybrid implementation could have for a LBM implementation that leverages a deep halo ghost cells pattern. In previous tuning studies conducted on Blue Gene/P, flat MPI and MPI/OpenMP programming models were shown to offer similar performance results for the LBM [165]. We found similar results as depicted in Fig. 5.12a, however, the hybrid implementation allows us to both increase the size of the fluid system that can be simulated and reduce the number of ghost cells because it reduces the number of domains of interest that the problem is broken into, thus directly reducing the number of ghost cells used. Recall that for any ghost cell depth n, the number of ghost cells in a simulation is equal to the area of the cross sections of the number of domains multiplied by $2n$.

This tradeoff also provides the ability to model larger fluid systems on smaller processor counts. For the results presented here, we used the maximum ratio from the previous studies, fluid dimension of 66 lattice points per processor for the D3Q19 model and 800 lattice points per processor for the D3Q39 model.

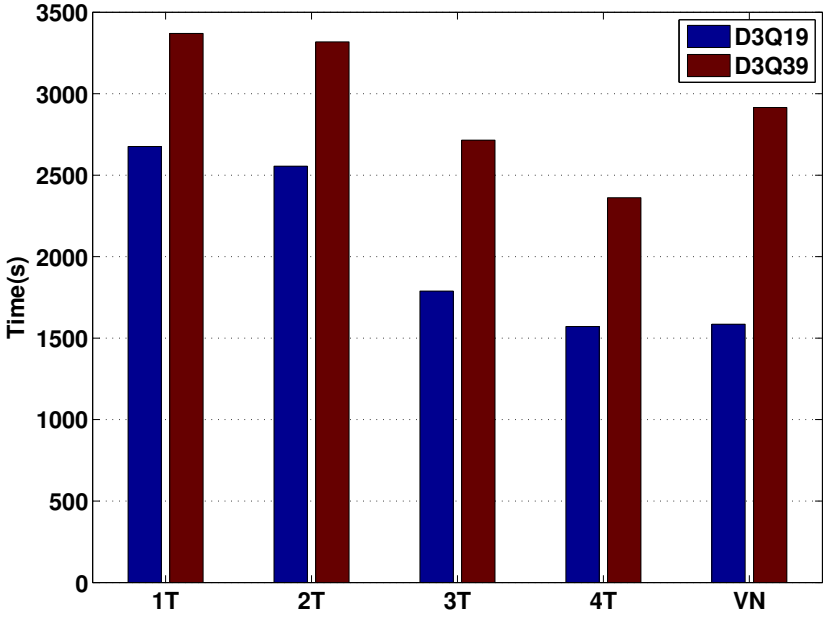
The first set of test was conducted on 32 nodes of Blue Gene/P exploring the use of 1,2,3 and 4 threads compared to results modeling the same fluid system but maxing out the MPI rank count through use of virtual node mode or four MPI processes per node. The simulations were run for ghost cell ranges 1-4 with the smallest runtime at each level being displayed to show maximum performance. Second, 16 nodes on Blue Gene/Q were used with a range of task and thread combinations shown in Fig.

5.12b.

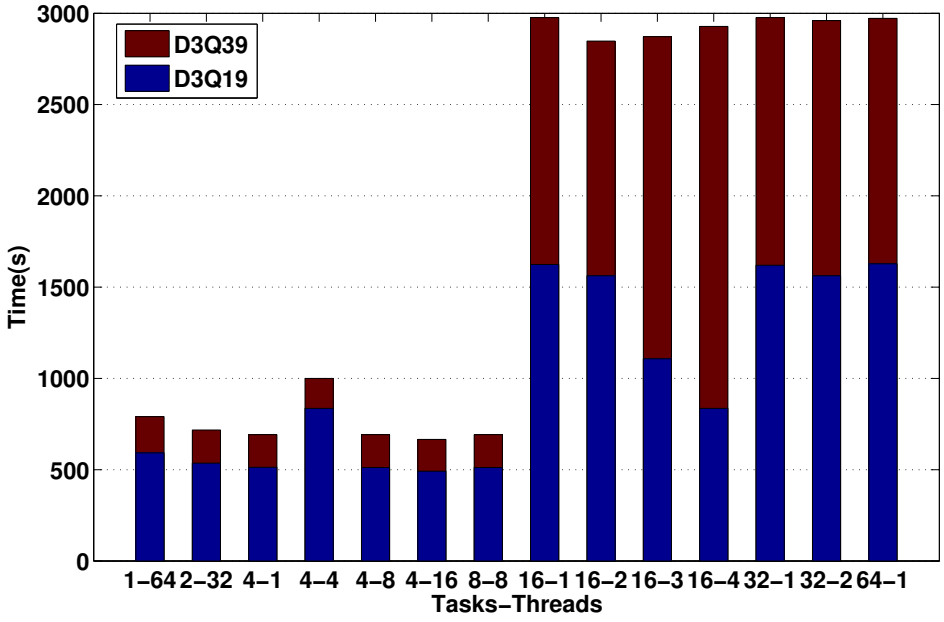
As shown in Fig. 5.12, threading improves the performance of both models for both platforms. The minimal runtime for the D3Q19 model on Blue Gene/P is approximately the same for the hybrid model with 4-threads or the flat MPI model run in virtual node mode. This result is consistent with the previous group's results. The interesting point here, is that for the D3Q39 model, the hybrid model with 4-threads with two ghost cells actually outperforms the virtual node mode case. This improvement is due to the reduction in ghost cell overhead as the number of border cells is decreased. There is a much bigger impact on the D3Q39 model as it not only requires more memory bandwidth for the extended velocities but also has to take into account the two-speed nature of the model, resulting in twice as much overhead as seen in the D3Q19 model. In regards to Blue Gene/Q, the naive expectation was that the optimal setup would have at least one task per processor and between one and four threads per task. Due to the aforementioned benefit of ghost cell reduction through the shared memory optimization, the optimal pairing of tasks and threads for the higher order model is actually four tasks per node with 16 threads assigned, as shown in Fig. 5.12b. This optimal pairing was true for both models.

5.7 Discussion

Modeling fluid flows beyond the Navier-Stokes regime has been a long posed challenge. With the extension of the LBM to the D3Q39 discrete velocity model, flows



(a) Blue Gene/P



(b) Blue Gene/Q

Figure 5.12: Impact of threading on both velocity model's performance. In each case here, the time of the minimal ghost cell implementation is shown.

at finite Kn are able to be accurately modeled and higher order kinetic moments recovered [171], allowing the accurate modeling of nanoscale flows such at those in the micro-vasculature or MEMS.

This extended model, however, introduces new computational challenges over previously studied LBM implementations. In this paper, the performance impact of the extension and methods to reduce this impact were explored. By maximizing data handling and streamlining the computation, results were produced demonstrating 92% and 83% of the predicted upper bound on performance on Blue Gene/P for the two models, consistent with the 30% demonstrated hardware efficiency for D3Q19 and 21% of peak for D3Q39. For Blue Gene/Q, the production results were at 85% and 79% of the predicted performance maximums, confirming strong correlation with the simple performance projection. An overall 3× improvement for Blue Gene/P and 7.5× improvement on Blue Gene/Q were exhibited.

It was shown that for both models, deeper levels of ghost cells proved beneficial as they minimized the overall number of messages being sent. The performance gain from deep level ghost cells, actually made the D3Q39 model with 4 threads outperform the single ghost cell implementation maxing out flat-MPI ranks on Blue Gene/P. Similarly, on Blue Gene/Q, the use of deep level ghost cells alongside the hybrid programming model produced efficient simulations of the extreme condition fluid flows. It was found that using a high level of threading per node resulted in maximal performance due to the ideal ratio of minimized communication due to ghost

cell use and minimized added ghost cell overhead due to threading.

Finally, it was demonstrated that the extended models are highly bandwidth limited, which poses limitations to the potential hardware efficiency when there is a greater imbalance between bandwidth and floating point capabilities. While significant runtime reduction is achieved on the Blue Gene/Q architecture, investigation into methods to alter the algorithm as to reduce the memory accesses per lattice update could increase the potential hardware efficiency on such systems. The simulations of fluid at extreme conditions present a real world example of an application where focus needs to be on memory bandwidth improvement over increased flop rate. While the work presented in this paper offers demonstrated performance nearing the upper bound predicted for this platform, it simultaneously highlights the need for more methods to bridge the gap between bandwidth and floating-point performance limitations.

6

Comparison of Simulation to *in vivo* Measurements

One concrete example is better than a mountain of prose.

– Freeman Dyson, *Scientific American [111]*

In this chapter, the results of the proposed blood flow simulation are evaluated and compared to *in vivo* measurements. The objective of this chapter is to assess the accuracy of the calculation of the pressure gradient through a moderate thoracic aortic co-arctation model. The focus is shifted to the pressure gradient in the aorta due to its ease of measurement. This quantity is a strong candidate for evaluating simulation as opposed to the aforementioned ESS that cannot currently be measured *in vivo* [80].

6.1 Motivation

The disease in question in this chapter is another CVD called the co-arctation of the aorta (CoA). CoA can pose a significant problem as the narrowing of the aorta can inhibit blood flow through the artery. CoA accounts for 8%-11% of congenital heart defects, affecting tens of thousands of patients annually in the western world [81]. While preventative actions such as balloon angioplasty or stent implantation can reduce the burden the heart by alleviating the pressure gradient [127], long-term results still reveal a decreased life expectancy associated with the disease. Studies have indicated that biomechanical properties could be a contributing factor ([81], [107]). The objective of personalized computer simulations of patients suffering from CoA is to further the understanding of hemodynamic flow patterns under both resting and exercised states to allow for a better understanding of likely sources of morbidity and an assessment of potential treatment outcomes [81].

Currently, surgical intervention is sought if the peak-to-peak systolic pressure gradient across the co-arcted vessel is measured at greater than 20 mmHg. This level was shown to be the cutoff for best results by physician intervention that incurred the least risk for the need of additional procedures [134]. This pressure gradient is not only determined by the size of the narrowing but also factors such as the flow rate of the fluid and the size, number, and placement of collateral vessels [127]. The physiological state of the patient can also contribute to an increase in the pressure gradient if, e.g., the patient is in an exercised state due to the associated elevation

in heart rate. When the patient is at rest, clinicians can easily measure the pressure gradient; however, this measurement is difficult to obtain under exercise conditions. This difficulty causes simulation to play a key role in determining the pressure gradient non-invasively.

In this chapter, the accuracy of the aforementioned HARVEY application is evaluated through comparison of measured patient data to the result of data informed simulation. In this case, the measurements from several medical imaging modalities are used to prescribe parameters like blood viscosity and inflow rates in order to get an estimate of the pressure gradient across the co-arcted aorta. The simulation of blood flow in the patient specific data involves the following five steps:

1. Acquisition of medical imaging data
2. Image segmentation to identify vessel geometry
3. Grid initialization
4. Flow simulation
5. Data analysis and simulation

The methods to obtain the medical imaging data, process it, and impose a regular grid and model the fluid flow using parameters provided from patient data will be discussed. Furthermore, a technique is introduced to use a patient specific inflow waveform to produce realistic pulsatile flow that upholds the measured flow distribution via velocity-imposed boundary conditions at the inlets and outlets. Finally, the

convergence of the pressure gradient is assessed alongside the overall accuracy of the simulation.

6.2 Geometry Data Acquisition and Segmentation

The patient data used in this chapter was provided by the STACOM CFD Challenge for the simulation of hemodynamics in a patient-specific aorta co-arctation model [127] and was discussed in detail in [81]. Gadolinium-enhanced MR angiography (MRA) was performed on the participant using a 1.5-T GE Sigma MRI scanner (GE Medical Systems, Milwaukee, WI). The participant was in the supine position and instructed to hold their breath during the acquisition period that lasted approximately $20s$. Simvascular software [21] was used to extract a 3D volumetric representation of the ascending aorta, arch, descending aorta, and upper branch vessels as shown in Fig. 6.1. MeshSim (Simmetrix, Clifton Park, NY) was used to generate a mesh file of the segmented MRA data. More details can be found in [83].

6.3 Initializing the Regular Simulation Grid

As discussed in Chapter ch:methodology, the LBM relies on a regular Cartesian grid being applied across the vessel geometry. The computational fluid dynamics equations are then solved at each grid point allowing the hydrodynamics properties to be deduced. To guarantee a proper initialization of the simulation grid, it is

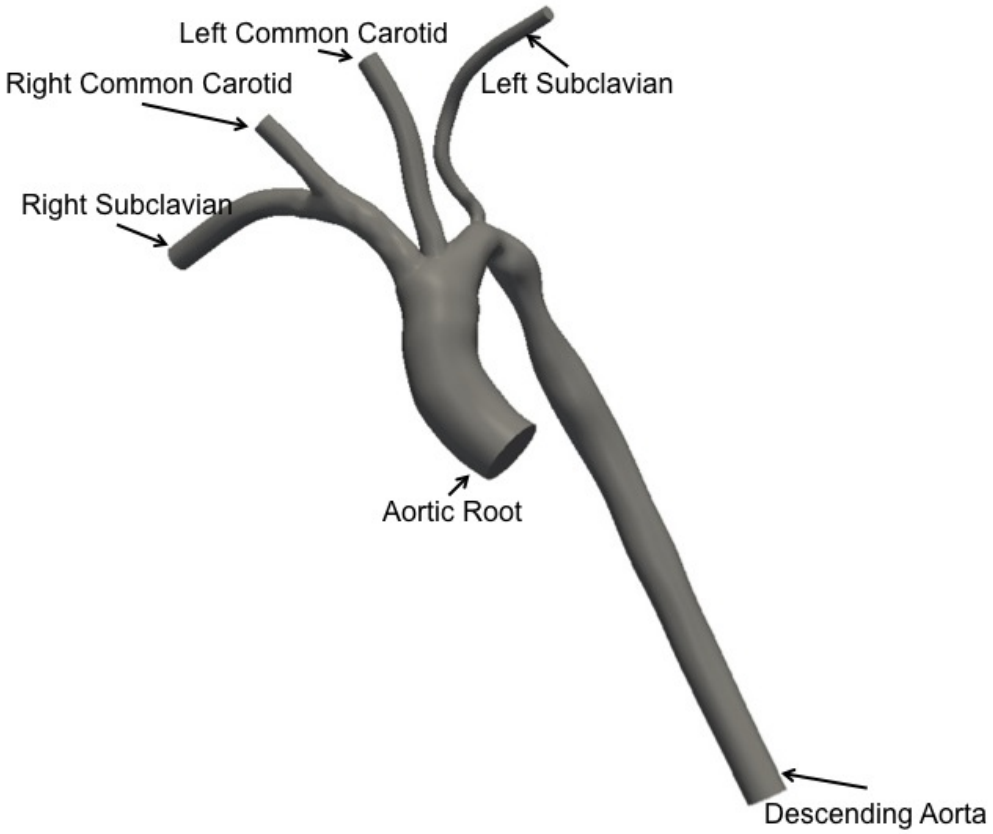


Figure 6.1: Patient specific aortic geometry acquired from the segmentation of MRA data. The specific vessels contained in the geometry are labeled.

required the patient's triangulated vessel geometry to be a closed, 2-manifold with no overlaps of interior volume. The process starts by computing the axis-aligned bounding box (AABB) of the input geometry offset by ε (I use $\varepsilon = 1.8c_i\Delta t$) on each side, then discretize the box's volume into a regular grid of targeted resolution. Note that ε is chosen to be slightly bigger than the length of the diagonal of a regular grid cube ($\varepsilon > \sqrt{3}c_i\Delta t$). With this choice, it can be guaranteed that every interior grid point has a neighbor not only in any 6-neighborhood but in any diagonal grid direction also.

Next, each grid point is classified to either be inside or outside of the given vessel geometry. Note that it would be prohibitively slow to run an inside-outside test for each individual grid point. The grid points falling inside the union of all sphere-sIpt triangles (the volume of a sphere-sIpt triangle is given by the union of all spheres of radius ε with centers on the triangle) and then "fill in" the inside-outside classification for all remaining grid points. More specifically, the method calls for the iteration over all vessel triangles: for each triangle, the computation of the grid points that overlap with its AABB offset by ε and then the validation against its sphere-sIot bounding volume. For the remaining grid points, the closest point on the triangle and – if the point hasn't been initialized yet or the current point is closer to the vessel geometry than the previously initialized one – is computed and the method will classify it as either inside or outside using the angle Iighted pseudo normal approach by Bærentzen and Aanæs [8]. Note that [8] guarantees a correct inside-outside classification for

points with respect to non-convex geometries provided that I know their closest points on the mesh. After a run through all triangles, I can guarantee to find the correct closest points for all grid points falling in the union of the sphere-sIpt triangles, hence, to correctly classify them using [8]. By using a radius of ε to setup the triangles, the method further guarantees the correct initialization of at least two inside grid points in any 6-neighborhood and diagonal grid direction within a distance ε of the vessel's boundary. Finally, the algorithm proceeds by looping over the three grid indices (that are monotonously decreasing or increasing in their respective dimension) to "fill in" the remaining grid points and classifying all grid points as inside if the loop index of the most inner loop has crossed the grid boundary an odd number of times.

This classification is then refined into wall, inlet, and outlet points by using the grid point's 6-neighborhood (if at least one of its six neighbors are classified as outside, there is a wall point) together with proximity information to inlet or outlet triangles. The remaining grid points are either "fluid" nodes (inside the wall), or "dead" (outside and not considered).

6.4 HARVEY

As mentioned, the LBM has proven to be a strong alternative to simulations derived from the Navier-Stokes equation of continuum mechanics and in this work, the developed LBM application, HARVEY, is used to assess the pressure and flow conditions in the aorta. A key advantage of the LBM is that macroscopic quantities

such as density are moments of the distribution function. This means that they can be calculated based on its summation and therefore are available entirely locally. In the study of CoA, the fluid pressure is particularly important. Pressure can be easily recovered through the ideal gas relation: $P = c_s^2 \rho$. This means that the value is available locally which is particularly advantageous as this means they do not require solving an expensive Poisson problem as in other CFD methods [100].

6.4.1 Boundary Conditions

Rigid walls are assumed throughout the coarse of the simulation and a no-slip boundary condition is imposed at the wall through the use of a full bounce-back method described in Chapter 2. The inflow and outflow boundary conditions are set by measured patient specific quantities obtained via a cardiac-gated, 2D, respiratory compensated, phase-contrast (PC) cine sequence with through-plane velocity encoding. Each scan lasted approximately 3 min while the subject breathed freely [127]. In this model, there is one inlet for the aorta and multiple outlets for each of the collateral vessels. The imposition of the boundary conditions is based on the knowledge of local flow profiles as provided by the measured patient-specific data.

Fig. 6.2 shows the result of the measurements of the ascending aortic flow with the red circles. In order to enable continuous flow throughout the heartbeat in the CFD simulation, the sum of sine functions was fit to the data, shown in Eq. 6.1, to determine the equation of the pulse. The fit procedure was performed in MatLab

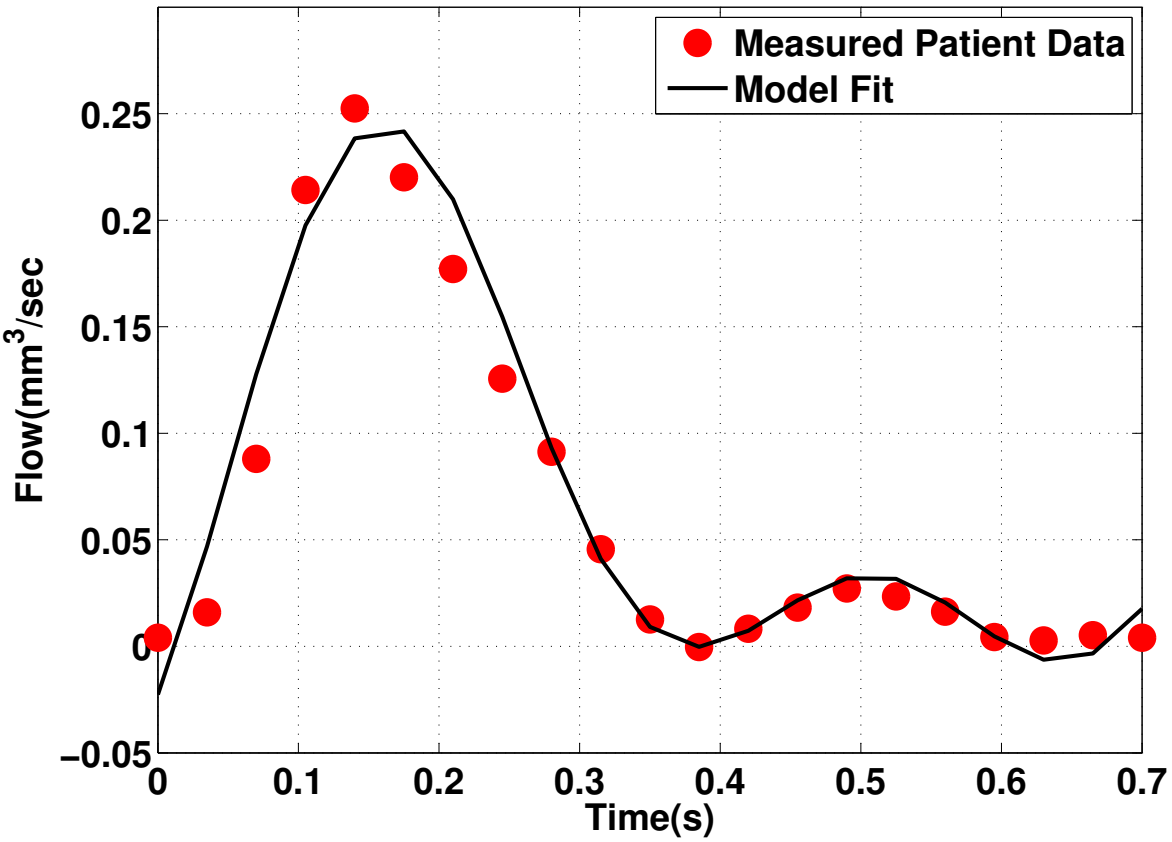


Figure 6.2: Fit to blood flow information acquired with PC-MRI. This fit is used for the simulation of only one cardiac cycle. A more complex and periodic fit would be leveraged when modeling multiple heartbeats.

Table 6.1: Coefficients for Eq. 6.1 with a 95% confidence bound.

Coefficent	**Value**
a1	0.6639
b1	0.211
c1	2.961
a2	0.08709
b2	16.75
c2	-0.5791
a3	0.1042
b3	11.88
c3	-1.129

using a non-linear least squares method and a trust region algorithm. The associated coefficients with a 95% confidence bound are shown in Table 6.1. Using a Pearson correlation, there was a statistically significant agreement between the phase contrast data and the equation derived velocity values ($R = 0.981, p < 7x10^{-15}$).

$$u(x) = a1 \cdot sin(b1 \cdot x + c1) + a2 \cdot sin(b2 \cdot x + c2) + a3 \cdot sin(b3 \cdot x + c3) \qquad (6.1)$$

A straightforward method of imposing a plug flow profile based on the flow rates above is employed. Any node defined as an inlet node has its velocity set based on the time point in the simulation ensuring proper pulsatile flow that matches the

Table 6.2: Percent of the inlet flow that is routed through each branch.

Location	**QIA**	**QLCCA**	**QLSA**	**QDAo**
% Ascending Aortic Flow	25.6	11.3	4.26	58.8

measured data.

For the outlet condition, additional PC-MRI planes were defined to obtain flow rates at the upper branch vessels and descending aorta. Table 6.2 shows these flow rates as a percentage of the aortic flow. QLCCA denotes the flow through the left common carotid artery, the QLSA is the left subclavian artery, and the QDAo is the flow through the descending aorta. QIA refers to the flow through the innominate artery. In order to determine flow going through the right subclavian and right common carotid outlet faces, the physiological condition is assumed that the pressure drop in each vessel is driven by the oxygen request from the tissues nourished by the vessel as in the empirical procedure described in ([100]). To this end, the flow splitting conditions of $\phi_1/\phi_2 = S_1/S_2$ in which ϕ_1 and ϕ_2 are employed to denote the outgoing flow rates and S_1 and S_2 the corresponding sectional areas. Coupling the incoming flow rate with the known flow splitting at each bifurcation asserts consistent outflow conditions. For the rest of the vessels, the measured outflow rates are imposed.

6.4.2 Memory Requirements

As previously discussed, the HARVEY simulation package is designed to handle complex geometries and to run large-scale simulations on high performance hardware resources. It has been developed from the ground up with parallel efficiency in mind to enable high-resolution runs. The mesh is Cartesian which enables straightforward data handling. It is written in C and uses MPI as the communication library. This code takes advantage of optimizations such as a) hand loop unrolling b) Single Instruction, Multiple Data (SIMD) intrinsics c) removal of redundant operations d) non-blocking communication discussed in Chapter 5 [115].

The domain is split such that each processor handles a set division of the Cartesian mesh. In HARVEY a double buffer approach is used in which a starting distribution of fluid particles is initialized for each lattice point. The advection step propagates the particles to adjacent lattice points and stores these values in a temporary distribution function. This is the data exchanged with the neighboring processors, and subsequently used for the collision step. The result of the collision step is then used to update the local portion of the original array containing the distribution function for each lattice point. In this manner, HARVEY acts as a typical stencil code that draws information from its neighbors, updates its local value, and pushes this data to the neighbors, however, the data accessed in the temporary data structure is from another phase space as well as from another lattice location.

This double buffer approach further increases the already large memory demand of

the simulation. In the case of this data set when simulated at a 200-micron resolution, there are 64,435 fluid voxels in a bounding box of 11,254,320 voxels. For each lattice point, there are two buffers that make up the bulk of the memory requirements. These buffers store the density data for each discrete velocity at each lattice point as a floating-point number. For a 200-micron resolution simulation, this requires at least three gigabytes of memory. While some commodity desktops may now be able to meet the memory needs for 200 microns, this becomes increasingly difficult at finer resolutions. For a 20-micron simulation, 3 terabytes of data are necessary. This is beyond the capabilities of traditional computers and requires the use of large-scale platforms such as the IBM Blue Gene/Q described in a following section, especially when simulating full heartbeats.

The second issue is the runtime for the simulation to complete. At high resolutions, the LBM requires rather small time steps on the order of 10^{-6} seconds resulting in the need for 700,000 time steps to complete one heartbeat. The computation of the solution of the LBM equations for each lattice point in a serial manner can take from hours to days at these resolutions. In order to drastically decrease the time to solution, a parallel implementation that allows us to simulate the full cardiac cycle in minutes is leveraged.

For the work in this challenge, the IBM Blue Gene/Q architecture was relied on. Similar to previous Blue Gene systems, it is built on a system-on-a-chip backbone and has expanded options for threading and memory access. The Blue Gene/Q system has

a 64-bit PowerPC processor operating at a 1.6 GHz frequency. Each node consists of 16 cores with 4 potential threads per core. There are capabilities for a 4-wide double precision FPU SIMD resulting in a 204.8 GFlop/s peak performance per node [67]. Memory per node is expanded to 16 gigabytes. In this work, 256 cores on 16 nodes of Blue Gene/Q are used for the simulations.

The velocity distribution at 0.14 seconds is shown in Fig. 6.3 for $100 - micron$ resolution simulation. The velocity is shown to vary between $0.0013m/s$ and $0.3m/s$. Regions subject to lower velocity fields occur at points of curvature in the wall whereas the highest fluid exhibiting the highest velocity is at the center of the arteries. As the region in the aorta subject to the narrowing, the velocity is shown to decrease above and below the co-arcted segment.

6.5 Results

The distribution function, f, at each lattice point is saved at a set time interval during the simulation allowing for post processing of the data to determine relevant macroscopic properties. It is during the post processing stage that the data is shifted from lattice units to SI units to allow for analysis of factor like fluid velocity, density, and pressure gradients. Only a subsample of time points are recorded and used for visualization and analysis. In this case, checkpoints are invoked every 20000 time steps. Paraview from Kitware is used to view the results [70].

To a fair approximation, the effect of non-Newtonian behavior of blood in the

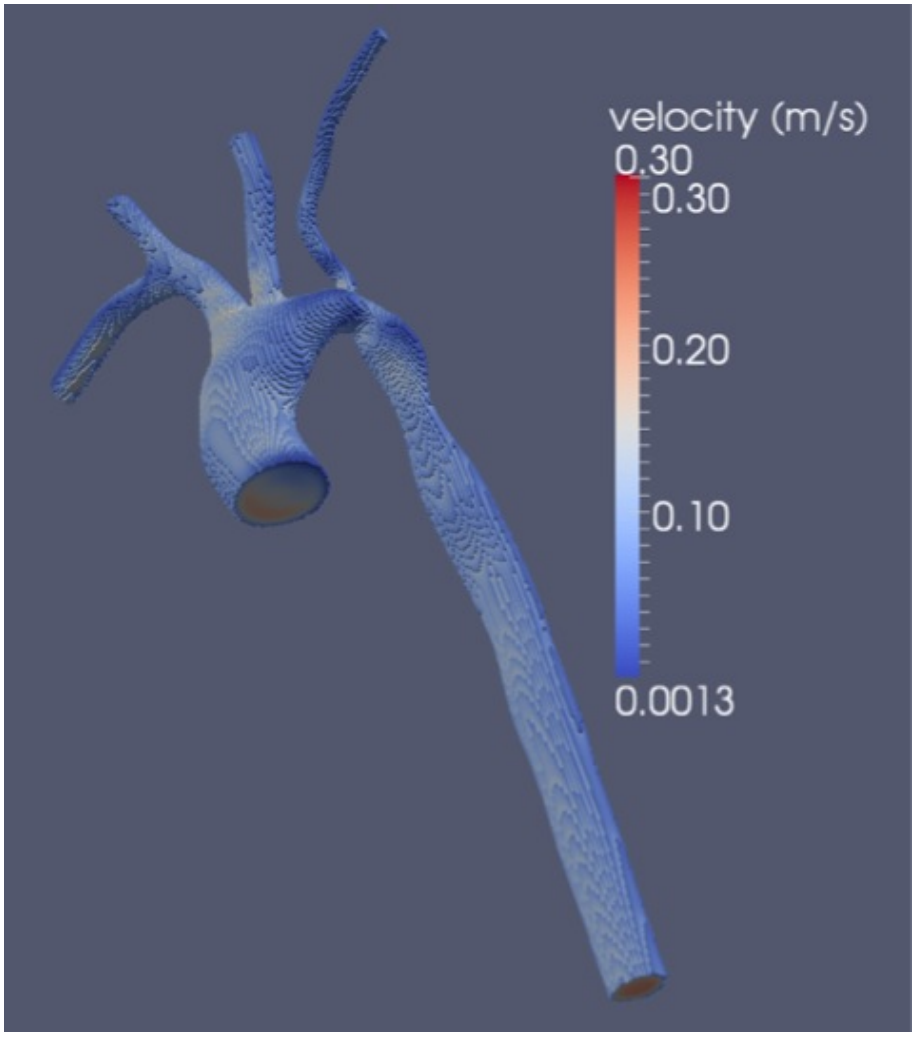


Figure 6.3: Mapping showing the velocity distribution at .14 seconds in a 100 micron resolution simulation.

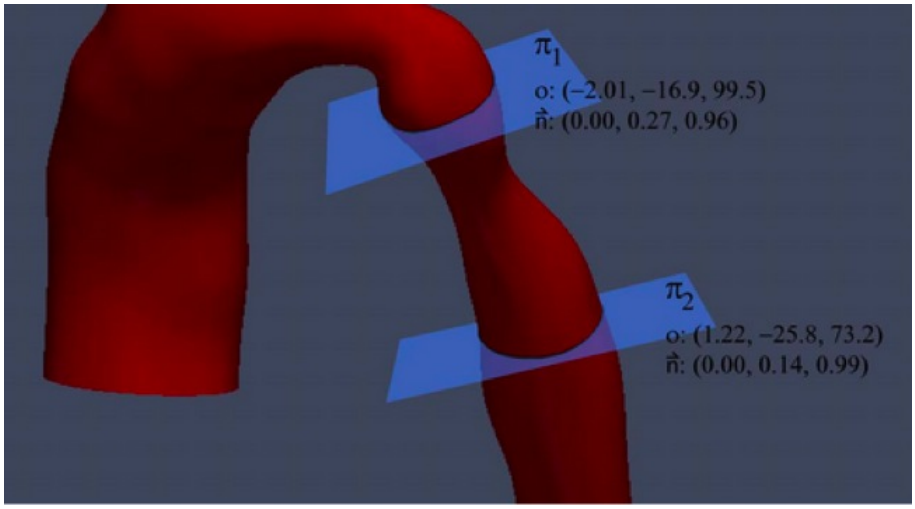


Figure 6.4: Location of the proximal and distal planes to the co-arctation site for reporting the pressure gradients [127].

aorta is negligible, and so Newtonian behavior of the blood is assumed. The physical density is set at .001 g/mm^3 and the dynamic viscosity is .004 gr/mm/s.

The mean pressure gradient between the upper and lower body is determined by taking the difference of the average pressure of the fluid in the plane at the proximal and distal locations as shown in Fig. 6.4 [46]. The results of simulations at three different resolutions are provided in Table 6.3. The pressure proximal to the co-arctation is measured at 113.1 mmHg (systolic) and 62.3 mmHg (diastolic), which correspond well to the measured values of 115 mmHg and 65 mmHg respectively.

Table 6.4 shows the simulation results for the peak and mean pressure differences as well as the flow splits and pressure in the AAo at a 20μm resolution.

Table 6.3: Mean pressure gradient at different mesh resolutions.

Resolution	**Pressure Gradient at Diastole**	**Pressure Gradient at Systole**
$200\mu m$	10.1 mmHg	12.2 mmHg
$100\mu m$	8.7 mmHg	10.9 mmHg
$50\mu m$	8.1 mmHg	10.4mmHg
$20\mu m$	8.2 mmHg	10.3 mmHg

Table 6.4: Full Results at $20\mu m$ resolution.

Peak pressure difference between Plane 1 and Plane 2	10.6 mmHg
Mean pressure difference between Plane 1 and Plane 2	9.2 mmHg
Flow splits in supra-aortic and DAo	40% and 60%
Pressure in AAo(Systolic/Diastolic)	10.3mmHg/8.2mmHg

6.6 Discussion

In this chapter, simulating blood flow in a patient specific geometry and estimating the pressure gradient in the aorta assess the accuracy of the method developed in prior sections. The system imposes a regular grid on the vessel geometry derived from the segmentation of MRA data and uses HARVEY, a lattice Boltzmann application, to model the blood flow through the arteries and to derive the fluid pressure gradients.

Simulations were undertaken to model the resting clinical blood pressure gradients. The results demonstrate an 8.2 mmHg pressure differential at diastole and 10.3 mmHg at systole. The average pressure difference across the entire cardiac cycle was estimated to be $9.2 mmHg$. The measured value was $12 mmHg$ giving an error of $2.8 mmHg$. This result improved on the simulations of previous CFD studies that used a finite element method to solve the Navier-Stokes equations and took into account wall elasticity [81]. This previous work exhibited an error of $5 mmHg$ and a mean pressure gradient estimation of $7 mmHg$. Furthermore, the results obtained by HARVEY were found to achieve the smallest error in regard to the average pressure gradient of any submission to the 2012 STACOM Computational Fluid Dynamics Challenge.

The results demonstrate the applicability of informing HARVEY with clinical imaging data to accurately assess the pressure gradient and hemodynamics quantities associated blood flow through a co-arcted aorta. Such a model can provide insight into the underlying mechanisms of the disease and steer potential treatment options.

7

Parallel in Time Approximation of the Lattice Boltzmann Method

Prediction is very difficult, especially about the future.

– Niels Bohr [40]

7.1 Motivation

Computational fluid dynamic (CFD) simulations of biological flows provide physicians the ability to identify regions of the circulatory system that are at risk for the development and progression of heart disease and have helped yield deep insights into the underlying mechanisms that experimental measurements alone could not have achieved (e.g. [99], [78], [149], [150], [160]). The disease trajectories that can be modeled, however, have been limited by the rate at which these simulations can

be performed. A significant challenge is to capture, *in silico*, functionally important biological events that typically occur on the timescales of anywhere from seconds to decades. It is important to not only understand the flow patterns over the coarse of one heartbeat, but in many instances the quantity of interest is the wall shear stress or fluid pressure across heartbeats. Furthermore, the target for these applications is to mode time domains that capture events such as plaque accumulation in the arterial wall. In such instances, simulations of weeks to years worth of simulated time would be required. Such long timescale simulations actually pose an arguably larger problem than modeling larger fluid systems for more moderate timescales. To model larger fluid systems, one can leverage the great array of prior art and exploitation of spatial parallelism that is not available when extending a simulation in time (c.f. [63], [113], [64], [30], [79], [165], [121]).

For many fluid simulations, the parallel efficiency saturates as soon as the size of the fluid domain drops below a certain limit. In these cases, the speedup gained from spatial decomposition will inevitably reach a plateau for a fixed problem size: increasing the spatial discretization will lead to saturation of parallel speedup when the number of lattice points per core becomes too small. After this point, any additional cores no longer improve the overall time to solution. Moreover, it is becoming more common that large simulations need to be completed on a short time scale in order to make a significant impact on scientific outcomes, making the wall-clock time a crucial component. If sufficient computational resources are available, decomposing

the problem in the time domain as well can assist in overcoming this strong scaling limit and significantly reduce the wall clock time. In this chapter, an efficient method for time parallelization of the lattice Boltzmann method is presented.

The focus will be on the parareal algorithm first introduced by Lions *et al.* [85], which consists of a coarse iterator calculated serially to initialize data at discrete time steps that is used as the input for a fine grid iterator that runs in parallel. The coarse solver is based on a larger time step and typically a coarser space discretization. Iterative refinement enables the compute-intensive fine iterator to be handled by the temporal parallelization. This method can be viewed as a predictor-corrector scheme in which the coarse solver is used to predict initial values for the full time range and these initial guesses are iteratively refined through application of the fine solver, with the refinement (correction step) completed in parallel. The algorithm consists of a series of these iterations completing when the results have converged within a certain tolerance and is advantageous only if convergence happens faster than it would have taken for the fine iterator to simply run serially. The parareal algorithm has been shown to be effective in a variety of fluid dynamics applications including the development of turbulent flow [50], [129], [128], [136]. Here a method to enable parallel-in-time simulation of the LBM for modeling fluid dynamics is introduced using a multigrid approach to define the coarse and fine iterators. This method therefore reduces the real time duration of simulations and lengthens the bound on time trajectories that can be modeled.

To gain further performance advantage, temporal decomposition is coupled with spatial decomposition and in this chapter is presented initial performance and error analysis results from introducing a novel space-time parallelism to the computational hemodynamics application, HARVEY [115]. The relationships between spatial and temporal domain decomposition that can lead to highly efficient models capable of taking advantage of next generation supercomputers are described. An adaptation of the parareal algorithm first introduced by Lions *et al.*, which combines independent coarse and fine resolutions in time to reduce the wall clock time of real time problems will be focused on [85]. The fine representation is more computationally expensive and is run in parallel on multiple time intervals to refine the result of that individual interval. The result for the coarse resolution is calculated serially and used to initialize the fine representation, which is in turn calculated in parallel. The coupling of the two iterators provides a predictor-corrector scheme that iteratively refines the initial values to the fine solver and completes the refinement (correction step) in parallel. The algorithm consists of a series of these iterations that reach completion when the results converge within a set tolerance. It is worth noting that this scheme is only advantageous when convergence occurs faster than a serial run of the fine iterator would take [50].

Introducing a space-time parallel scheme into a lattice Boltzmann code such as HARVEY is not straightforward and requires computational and algorithmic developments to address challenges related to the accuracy, scalability, and stability of

coupling these two fluid representations. A main contribution in this chapter is the development of a multi-level space-time coupling (MSTC) that enforces a hierarchical decomposition of the default communicator and allows seamless communication between the spatial and temporal decompositions.

This chapter is organized as follows: in Section 7.5 the general formulation of the parareal algorithm is given. The definition of the coarse and fine iterators and necessary coupling mechanism is presented in Section 7.6, where there is also a discussion of techniques for conserving macroscopic quantities like viscosity across the refinement levels, and the expected performance model. This is followed with a discussion of the method for extending the model to include spatial decomposition as well as temporal.

In Section 7.9, the results of LBM with temporal parallelization are presented, showing that up to a 32× increase in speed can be achieved for a model system consisting of a cylinder with conditions for laminar flow. Finally, the first space-time parallel simulation of cardiovascular flows in realistic human arterial geometries derived from clinical imaging data is presented. The ability of our method to not only efficiently produces accurate results, but to recover time-dependent phenomena like the flow rate imposed by a heartbeat is demonstrated. Finally it is proven that space-time parallelization can help applications make better use of the large core counts available on next generation systems and overcome the intrinsic strong scaling limit to achieve a shorter time to solution than obtained with spatial parallelization

alone.

7.2 Related Work

Parallel-in-time methods have been investigated as ways to go beyond the strong-scaling limit of many applications. If the hardware is available, the combination of a coarse and fine solver can converge and enable a shorter time to solution for models of interest. The parareal algorithm was first proposed in 2001 [85] and solidified in the predictor-corrector format shortly after (e.g. [10], [9]). It converges with the same accuracy as would be achieved with the expensive or fine iterator. Recently, alternate parallel-in-time methods have been propose such as the parallel implicit time-integration algorithms (PITA) in which parallelizes the time-loop of a time-dependent PDE solver without impacting the serial compo (c.f. [44],[45]). Other studies use intertwining the iterations of parareal with the spectral deferred corrections, the Parallel Full Approximation Scheme in Space and Time (PFASST) ([102], [56]). Speck *et al.* demonstrated a speedup of 8× in addition to the spatial speedup on up to 294,912 cores [141].

In this chapter, the focus is on adapting the parareal algorithm to enable space-time parallel simulation of computational fluid dynamics (CFD). Previous work has shown the potential for such algorithms in CFD applications like in recovering time dependent behavior such as the development of turbulent flow [128], however, most of the literature emphasizes the algorithmic aspects of the time-parallel schemes and

there are only a few examples of studies into the efficiency of space-time coupling. There has been some research into the use of the parareal algorithm combined with spatial decomposition for a Navier-Stokes solver on up to 2,048 cores modeling flow passed a cylinder [50], but there are no studies that we are aware of investigating the coupling of spatial and temporal parallelism at larger scales for CFD for complex flow in a real 3 dimensional problem.

7.3 Spatial Scaling Limit of the Lattice Boltzmann Method

While the LBM has been shown to scale with high efficiency up to 294,912 cores [113], there is a limit to the strong scaling potential when using only spatial decomposition. As more cores are used, fewer and fewer fluid nodes are allocated per code. As this ratio diminishes, the cost of the internode communication starts to overwhelm the runtime and reduce the scaling efficiency. In this work, an even spatial decomposition was employed in which the bounding box of the fluid was broken up into small cubes of equal sizes by evenly splitting each dimension. Fig. 7.1 shows the speedup achieved on up to 32,768 cores of the IBM Blue Gene/P supercomputer at Argonne National Laboratory. The blue line indicates the strong speed up for a large fluid system consisting of 500 million fluid nodes. The green line shows the result for a medium sized system of 100 million fluid nodes and the red line denotes a small

system of 10 million fluid nodes. As shown, for fixed-size fluid systems, beyond a certain point, adding more cores can actually slow down the simulation. As the number of fluid nodes per core drops below 5000, the time spent in communication starts to overwhelm the computation resulting in lower overall speedup. Fig. 7.2 highlights this drop off showing a decrease in parallel efficiency for three different LBM codes. All of the scaling studies were completed on IBM Blue Gene/P supercomputers. The data for the second two applications were obtained from [113] and [30] respectively. While the second two codes included red blood cell modeling as well as the CFD component, all three demonstrate the same overall decline in efficiency corresponding to the number of fluid nodes per core. This is the intrinsic spatial scaling limit and will serve as the baseline for our parallel space-time simulations.

7.4 Parameters

The following parameters are used to setup the coarse-grained and fine-grained solvers for applying the parareal algorithm to the LBM.

T = overall time to be simulated

C = (ν/ν_o) where ν is the viscosity of the fluid in dimensionless lattice Boltzmann units and ν_o is the viscosity in terms of physical units (m^2/s).

γ_G = cost of coarse iterator

γ_F = cost of fine iterator

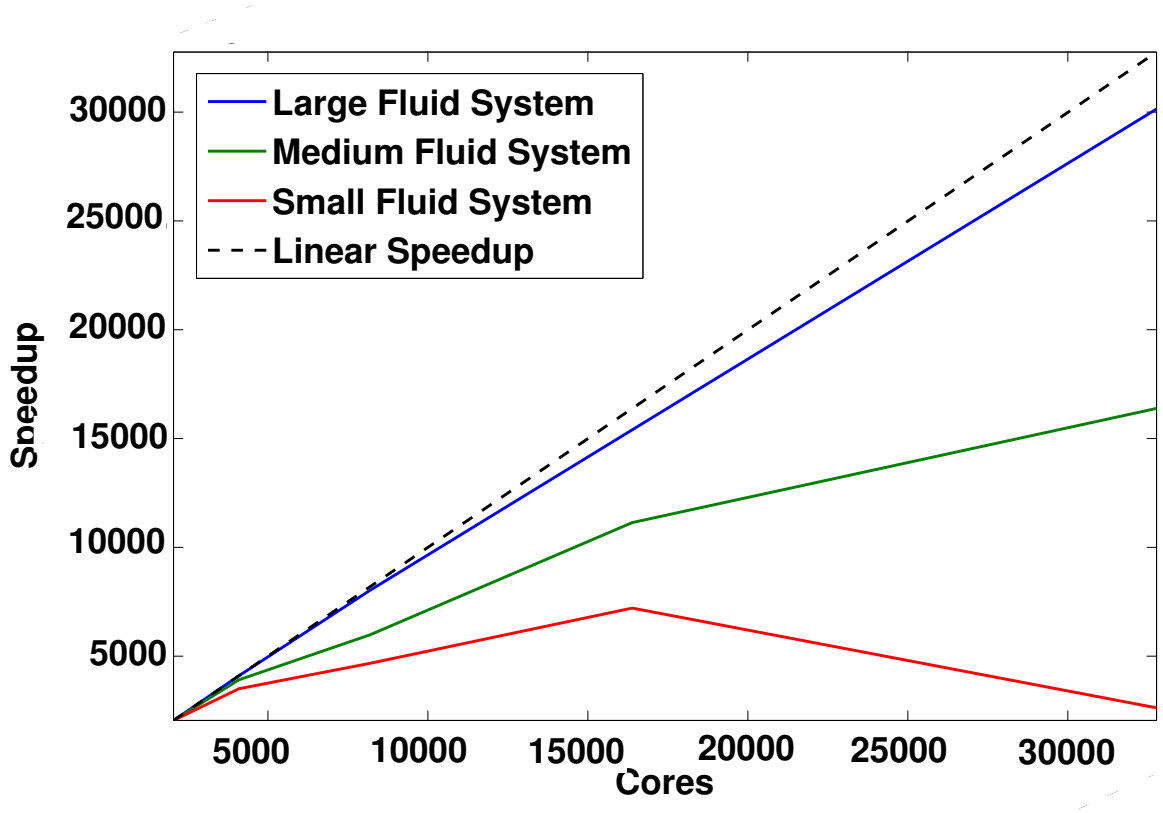


Figure 7.1: Speedup of LBM simulations using HARVEY for a range of fluid system sizes on up to 32,768 cores of the IBM Blue Gene/P supercomputer.

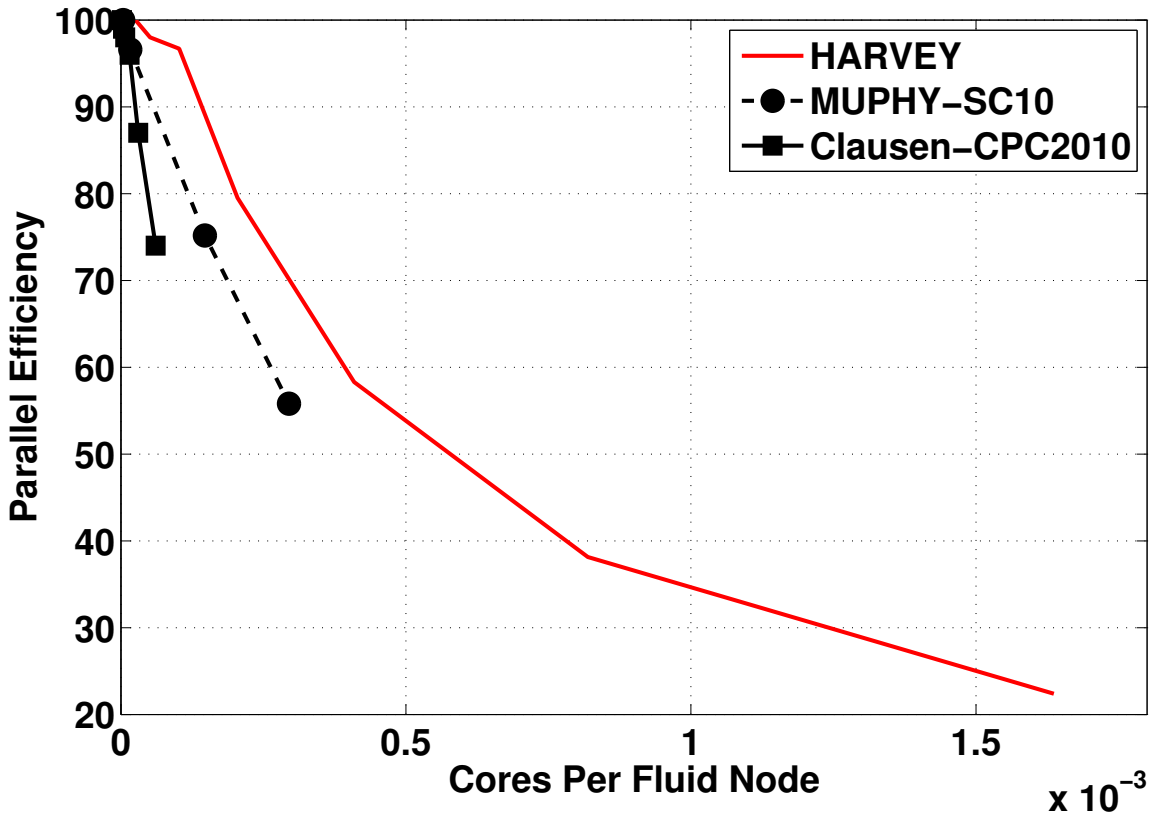


Figure 7.2: Parallel efficiency in terms of cores per fluid node for three different lattice Boltzmann codes. HARVEY is the application presented here. The other two codes include red blood models and the scaling studies were also completed on IBM Blue Gene/P supercomputers (c.f. [113], [30]).

lx = length in x-dimension of fluid system

ly = length in y-dimension of fluid system

lz = length in z-dimension of fluid system

dx_c = coarse grid size

dx_f = fine grid size

dt_c = time step for coarse iterator

dt_f = time step for fine iterator

ts_c = number of time steps to complete coarse iterator

ts_f = number of time steps to complete fine iterator

τ = wall clock time to simulate flow for one time step at one grid point

num_C = number of grid points in coarse grid

num_F = number of grid points in fine grid

7.5 Parareal algorithm

Lions, Maday, and Turinici (LMT) [85] first proposed the parareal algorithm as a method to decompose an ODE solver in the time domain. The method enables more effective utilization of high performance parallel systems by numerically solving a time dependent system in a shorter overall wall-clock time [15]. Temporal decomposition is achieved through a predictor-corrector scheme consisting of a prediction step, denoted by $\mathcal{G}$, that provides a coarse approximation and a correction step, denoted by $\mathcal{F}$, that contributes a more accurate but more computationally expensive

solution. This method alternates between the serial update of initial conditions with $\mathcal{G}$ followed by the parallel application of $\mathcal{F}$ until the corrected values converge within a predetermined tolerance, ε. By computing $\mathcal{F}$ in parallel, a shorter overall runtime is obtained [10]. Inherent to the method is the need for $\mathcal{G}$ to be less computationally expensive than $\mathcal{F}$. The disparity between these two iterators determines the potential speedup of the time parallelization.

In the parareal algorithm, the time interval to be modeled by the simulation is divided into N separate intervals of equal size, $[t_{n-1}, t_n], n = 1 \dots N$, with n denoting the n^{th} time step. For simulations that are split entirely in the temporal domain, N is set to the number of processors in the system. A different processor is responsible for each of these discrete time intervals. For each interval, a succession of approximations denoted U_{n+1}^K are calculated where K is the parareal iteration number. For each parareal iteration, K, the following steps are taken [129], [136]:

0. $(K = 0)$ The coarse prediction step $\mathcal{G}$ is first applied serially on all processors to calculate U_n^0 for the full simulation time, $0 \dots, t_N$.

1. $(K > 0)$ Using the initial value U_n^0, each processor can calculate $\mathcal{F}$ in parallel on its respective time interval, t_{n-1} to t_n. The result is then propagated to the next processor in line.

2. The serial correction step is calculated through:

$$U_{n+1}^{K+1} = \mathcal{G}(t_{n+1}, t_n, U_n^{K+1}) + \mathcal{F}(t_{n+1}, t_n, U_n^K) - \mathcal{G}(t_{n+1}, t_n, U_n^K) \tag{7.1}$$

Note that the second and third terms on the right-hand side of this expression have been obtained in previous iterations and steps.

3. Convergence is checked through the condition $\mid U_n^K - U_n^{K-1} \mid < \varepsilon$ where ε is the predetermined tolerance value. If the difference between the solutions for two successive parareal K iterations is smaller than ε for all time intervals, the parareal cycle completes.

In the work described in this chapter, a pipelined implementation of the parareal algorithm is used in which the fine approximation commences as soon as the coarse approximation is available for the time interval to be calculated by each processor, rather than waiting for all processors to complete the $\mathcal{G}$ calculation [102], [77]. This way, every processor computes each step of the parareal algorithm as soon as possible, removing the need for processors to remain idle while waiting for all of the other processors to finish calculating $\mathcal{G}$ [102]. Fig. 7.3 illustrates the method in a diagram for $K = 3$. The blue arrows indicate the wall-clock time (vertical component) for each step of the coarse iterator. In the example shown here, each processor handles the duration of one coarse time step (horizontal component of blue arrow). The red arrows show the wall-clock time (vertical component) required for the fine iterator, as described in Step 1. As will be discussed in more detail later, the fine iterator requires more time steps and a larger number of grid points at each time step, which implies a higher computational cost for each fine step, as indicated by the magnitude and the slope of the red arrows relative to the blue arrows in Fig. 7.3. The green

arrows indicate the propagation of data needed for Eq. (7.1) in the serial correction described in Step 2. Communication between processors occurs in the yellow shaded regions and is presumed to have negligible computational cost. The convergence tests are conducted at each black circle by comparing the result at that circle with that of the circle vertically below it, as described by Step 3.

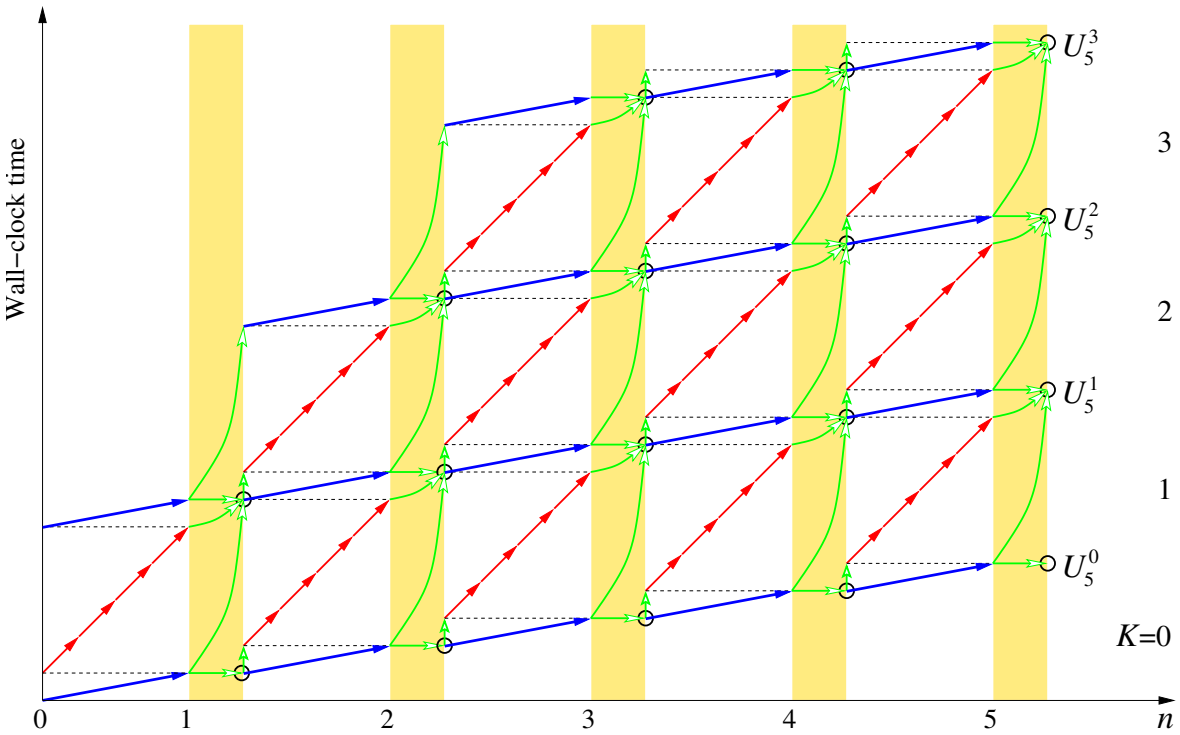


Figure 7.3: Computational cost of the pipelined parareal method for K=3: each processor is handling the time duration covered by one coarse time step, as shown along the horizontal axis. The cost of $\mathcal{G}$ in terms of wall-clock seconds per step is shown by the blue arrows and the cost of $\mathcal{F}$ is shown by the red arrows. The green arrows indicate the propagation of the data used in the correction step and the shaded regions correspond to communication between processors. At each black circle, converge tests are conducted. The corresponding speedup for each K value is shown on the right. The magenta bars indicate speedup given 10 processors and the aqua bars show the speedup given 100 processors.

7.6 Adaptation for the Lattice Boltzmann Method

our approach to enabling parallel-in-time simulation of the lattice Boltzmann method is defined next. To this end, the coarse and fine iterators are defined through use of grids at different resolutions. As discussed in Section 2.1, the LBM relies on discrete particles moving and interacting on a regular uniform lattice. Different resolutions of the simulation are achieved through the use of different size grid discretization levels, which define the coarse and fine grid iterators. The use of regular grids in which the local connectivity remains unchanged but the lattice spacing itself is refined is not new to the LBM. Typically, grids of different spacing have been used to produce multiscale models with finer resolution at specific regions in the fluid domain, such as at the walls or boundaries. Research has also shown the success of locally embedded grids for multigrid schemes modeling high Reynolds number flows around objects (c.f. [145], [47], [49]). our definition of the two different grid levels to enable temporal parallelism builds on previous work using mesh refinement [11]. Other factors that need to be taken into account include the time stepping mechanism and continuity of the kinematic velocity across the grids. These are each discussed in detail in the following.

Filippova and Hanel [47] developed a node-based approach for mesh refinement that relies on a hierarchical refinement of coarse and fine grids for the LBM. This work was extended to enable the use of fewer time steps on refined grids without any degradation in accuracy in space or time [48]. In the present approach, such

a two-level hierarchical grid refinement strategy in which a coarse grid covers the entire spatial domain and a finer grid is superimposed is leveraged. Grid refinement is implemented by dividing the spatial discretization by a refinement factor m. Fig. 7.4 shows a refinement in which the coarse grid is half the resolution of the fine grid, with $m = 2$. The red dashed lines indicate the placement of the fine grid while the blue lines show the coarse grid. Fig. 7.4 also depicts the lattice points encompassed by the simulation to emphasize the overlapping node-based method. The fine iterator uses all grid points (colored blue and red) and the coarse iterator only simulates the flow at the blue lattice points.

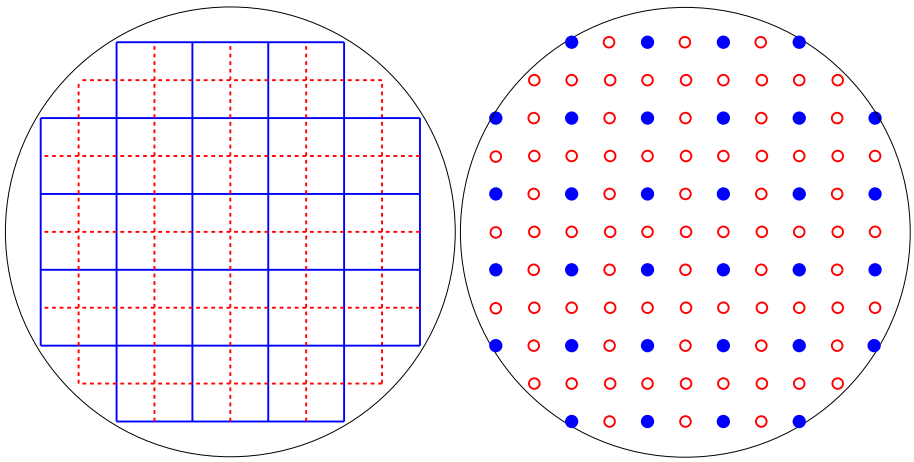


Figure 7.4: Left panel: a two level grid where the fine grid is represented with red dashed lines and the coarse grid with solid lines. The fine grid in this example has twice the resolution of the coarse grid, with $m = 2$. Right panel: the lattice points highlighted to demonstrate the overlapping method used. The fine iterator applies to each grid point, both red and blue, whereas the coarse iterator applies only to the blue ones.

In contrast to conventional multigrid methods, the entire spatial domain is covered with the grids for the two different iterators and the fluid motion on each is

modeled separately. Furthermore, the key challenge for multigrid and adaptive mesh refinement implementations is the handling of advection from one grid resolution to the other. However, due to the fact that the grids span the full spatial system, fluid particles are not streaming from one grid spacing to another between time steps for the introduced time-parallel technique. Conversely, the hurdle to overcome lies in the propagation of initial conditions from one grid resolution to the other and subsequently the coupling between the grids.

There are two key operations needed to address this challenge and coordinate between grid hierarchies: coarsening and interpolating. In order to transfer information from the fine grid to the coarse grid, a coarsening process must take place in which the distribution function at each velocity for the fine grid is averaged to pass the data to the coarse grid. This operates on conserved values and introduces no further truncation error. In order to move in the opposite direction, an interpolation method is required to transfer information from the coarse grid to the fine grid, that is, to fill in the data for the new lattice sites required by the fine iterator.

Within the two grid schemes, the time steps required for a simulation with each grid differ in size, resulting in a different number of time steps to be modeled by each iterator. In addition, each iterator relies on different relaxation parameters. Specifically, the modeling of fluid motion on the coarse grid requires the use of large time steps while modeling of the fluid motion on a fine grid relies on many smaller time steps. The size of a time step, dt, is related directly to the square of the grid

resolution, dx in the LBM: $dt = dx^2(\nu/\nu_o)$ where ν is the viscosity of the fluid in dimensionless lattice Boltzmann units and ν_o is the viscosity in terms of physical units (m^2/s). The number of time steps calculated by the coarse and fine operators is determined by the size of each respective time step.

A local second-order refinement solution for coupling between the grids that relies on different relaxation parameters and lattice spacing to transition between the grids as described in [47] was developed to ensure stable coupling. The kinematic viscosity in the LBM is defined by $v = c_s^2(\frac{1}{\omega} - \frac{dt}{2})$ where dx is the lattice spacing and ω is in dimensionless lattice Boltzmann units. For multigrid methods, the relaxation parameter ω in Eq. (5.1) must be rescaled to keep the viscosity constant across both the coarse and fine grids [47] . Here, the modification introduced by Dupuis *et al.* [37] that removes the potential singularity from the initial formulation was used that redefines ω as:

$$\omega_f = \frac{\Delta x_c}{\Delta x_f}\left(\omega_c - \frac{1}{2}\right) + \frac{1}{2},$$

where Δx_c and Δx_f are the spatial discretization size for the coarse and fine grids, respectively, and ω_c and ω_f are the corresponding relaxation parameters. This ensures that the simulation remains stable and introduces an upper bound on N by requiring ω_c to remain close to 2 and ω_f to be greater than 1 [47]. The use of this modified definition of ω for each grid imposes a finite limit on the disparity between the two iterator's resolutions.

The parareal algorithm applied to the LBM involves the following steps in which

each processor, n, computes the simulation for time domain $[t_{n-1}, t_n]$:

0. ($K = 0$) Initialize the coarse grid with a serial LBM simulation for the time domain $[t_{n-1}, t_n]$ for each processor; interpolate to initialize the fine iterator.

1. ($K = K+1$) Each processor separately applies $\mathcal{F}$ starting with the initial values provided by the previous iteration for t_{n-1} to determine the distribution function at t_n for its respective interval of time. This is completed in parallel and the result is shared with the next processor. As a serial process, $\mathcal{G}$ is applied.

2. The correction to $\mathcal{F}$ is calculated via Eq. (7.1); the result is coarsened to update the initial conditions for $\mathcal{G}$ and propagated to the next processor.

3. Convergence is checked: if all intervals of time have converged, exit the cycle; else, return to Step 1.

The key components to this scheme are the steps required to link the two different grid resolution levels in the *interpolate* and *coarsen* steps. There are a few important points to note: errors are introduced near the walls at boundaries as the initial conditions are taken from the coarse grid. When a boundary is interpolated, fluid cells are introduced that did not exist in the coarse grid and have no fluid cell from which to draw initial data. In this case, the average of all surrounding fluid cells is used to initialize the distribution function at this point. With each K-cycle, this approximation is updated from previous results on the fine grid, reducing the error at the boundaries over time. Convergence is defined by requiring the average relative

error of the solution in the K^{th} iteration of f_i to differ from the solution of the $(K+1)^{th}$ iteration of f_i by an amount less than a prescribed tolerance ε [136].

7.7 Coupled Spatial and Temporal Decomposition

To optimally leverage the available hardware of massively parallel supercomputers and ultimately minimize the overall runtime of the applications in question, it is posited that leveraging both spatial and temporal domain decomposition is needed for fixed-size problems. A general approach is introduced for coupling these decomposition strategies through a Multi-level Space-Time Interface. This is similar to communicator breakdown introduced by Grinberg *et al.* in [64] to enable the coupling between physical models. The key advantage of the MSTC architecture is the hierarchical decomposition of the default World communicator into sub-communicators to enable efficient coupling of parallel decomposition in both time and space. This decomposition is handled by splitting the World communicator into N different sub-communicators to handle the different time intervals of equal size. This is handled in a topology aware manner assigning cores that are physically near each other to the same Tier 2 ($T2$) group of cores. If no temporal decomposition is being used, all cores are assigned to the same $T2$ group. These groups are further subdivided through spatial decomposition strategies defining Tier 3, $T3$, non-overlapping groups. The core kernel being modeled in this framework would be solved within $T3$ groups.

Fig. 7.5 shows the layout of the MSTC. At $T2$, new communicators are intro-

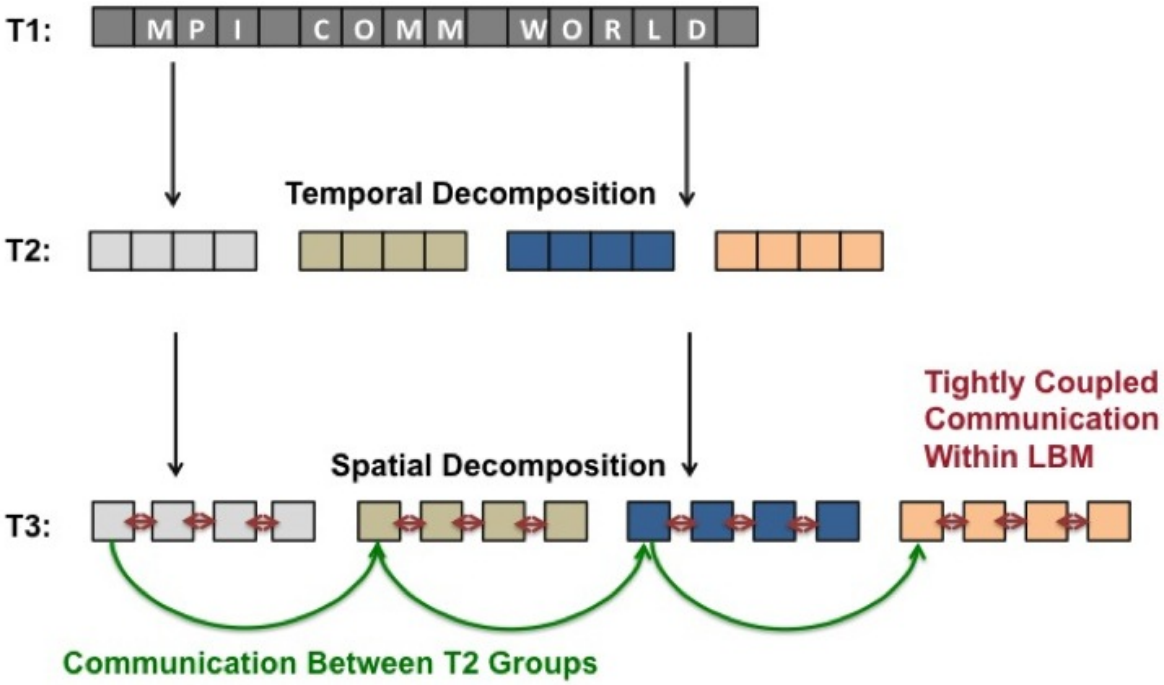


Figure 7.5: Multi-level Space-Time Interface breakdown. For Tier 2 ($T2$), the World communicator is broken into separate communicators handling temporal intervals. For Tier 3 ($T3$), each $T2$ group is broken up spatially. Coarse and fine solvers run across $T3$ groups. The red arrows indicate the tightly coupled message passing with the LBM and the dashed lines indicate the communication between $T2$ groups for the one core of each.

duced splitting the cores into groups handling each time interval and at $T3$, these are each further decomposed to handle specific spatial regions. The arrows in $T3$ give a view of the communication patterns involved with MSTC. The red bi-directional arrows indicate the tightly coupled interaction between cores in the same $T3$ group. This is defined by the kernel in question and can involve either point-to-point or global communication across all cores within that group. The kernel here defines both the coarse and fine solvers that will be executed within each $T3$ group. The dashed arrows indicate the point-to-point communication between $T2$ groups. These provide the interaction between temporal intervals in which the core handling a set spatial region for a time interval will inform the estimation of the corresponding core that handles the same spatial region in the next time interval. This is implemented through point-to-point communication between cores that have the same $T3$ ranks. The communication shown here simply indicates messages for cores of rank 0 in each, but similar message patterns occur for each rank. The bulk of the computational time for the simulation actually occurs within the $T3$ groups themselves and the message passing between $T2$ groups only occurs N times. As the calculations in each $T3$ group need to proceed in sync, blocking receive protocols are leveraged to ensure this. The pipelined nature of the parareal algorithm though allows the use of non-blocking sends to minimize communication overhead.

The $T3$ groups are implemented as input variables to the coarse and fine solvers, so one could envision future methods where one would redefine the communicators

throughout the coarse of the simulation allowing processors that would remain idle from K=0 time interval T communicators redefining the late communicators, joining them and providing further spatial scaling (up to the limit of course).

To define the coarse and fine solvers, a two-level hierarchical refinement of coarse and fine grids is used in which a coarse grid covers the entire spatial domain and a finer grid is superimposed [47]. The coarse iterator models fluid moving using the coarse grid and similarly the fine iterator is solved across the fine grid. As before, the entire spatial domain is covered with both the coarse grid and an overlapping superimposed finer grid. Each iterator models the fluid on its corresponding grid points separately for the entire spatial domain. To ensure accuracy and minimize communication overhead, the coarse and fine grid points for the same spatial region are mapped to the same core.

In the LBM, the resolution of the imposed grid spacing determines the time step size and contributes to the calculation of the kinematic viscosity of the fluid. The size of a time step, dt, is related directly to the square of the grid resolution, dx in the LBM: $dt = dx^2(\nu/\nu_o)$ where ν is the viscosity of the fluid in dimensionless lattice Boltzmann units and ν_o is the viscosity in terms of physical units (m^2/s). The kinematic viscosity of the fluid is determined as $v = (2/\omega - 1)dx \cdot (c/6)$ where dx is the lattice spacing and $c = dx/dt$. It is the impact on these two quantities that makes the adaptation of parareal for the LBM a challenging task and not a straightforward implementation. The coupling for the fluid model on the coarse and

fine grid introduces potential accuracy and stability issues to the method that need to be addressed.

As in the initial studies applying simply temporal decomposition to laminar flow in a tube, a local second-order refinement solution for coupling between the grids that relies on different relaxation parameters and lattice spacing to transition between the grids was used [117]. The relaxation parameter ω in Eq. (5.1) must be rescaled to keep the viscosity constant across both the coarse and fine grids to ensure a stable coupling mechanism [47] . Again ω is redefined as:

$$\omega_f = \frac{\Delta x_c}{\Delta x_f}\left(\omega_c - \frac{1}{2}\right) + \frac{1}{2},$$

where Δx_c and Δx_f are the spatial discretization size for the coarse and fine grids, respectively, and ω_c and ω_f are the corresponding relaxation parameters [37]. This both addresses the stability concern and imposes an upper bound on N by requiring ω_c to remain close to 2 and ω_f to be greater than 1 [47]. Moreover, the use of this modified definition of ω for each grid imposes a finite limit on the disparity between the two iterator's resolutions.

From here MSTC can be adapted to this locally embedded version of the LBM using the following steps:

0. $(K = 0)$

   Initialize $T2$ and $T3$ groups.

   Define neighbors between $T2$ groups.

In each $T3$ group, initialize the coarse estimation with serial LBM simulation for the time domain $[t_{n-1}, t_n]$ for the assigned spatial region.

Interpolate to initialize the fine solver.

For $(K = K + 1)$

1. $\mathcal{F}$ is across all $T3$ groups starting with the initial values provided by the previous iteration for t_{n-1} to determine the distribution function at t_n for its respective interval of time and space.

2. In each $T3$, $\mathcal{G}$ is applied. The correction to $\mathcal{F}$ is calculated via Eq. (7.1).

3. This result is coarsened and propagated between $T2$ groups to update the initial conditions for $\mathcal{G}$.

4. Convergence is checked: if all intervals of time have converged, exit the cycle; else, return to Step 1.

The key components to this scheme are the steps required to link the two different grid resolution levels in the *coarsening* and the *interpolation* steps. On each core, a coarsening function in which the distribution function at each velocity for the fine grid is averaged and rescaled is required to move data between the two grid levels. This operates on conserved values and introduces no further truncation error. Similarly, an interpolation method is required to address the new lattice sites requires by the

fine iterator. It is in both of these functions that the rescaling of ω is employed to maintain a constant kinematic viscosity across the space-time decompositions.

To enable the pipeline approach in a space-time coupling, non-blocking sending of messages is employed but blocking message passing by the receivers is required. This prevents cores from within one subcommunicator to get out of sync. All cores in the subcommunicator must handle the advection and collision within one step sequentially as the following step relies on data from its nearest neighbors from the previous step.

It is worth noting that even as larger fluid systems are modeled, the overhead of the message passing between $T2$ groups will remain constant. This is due to the fact that the gain from spatial speedup should always be maximized before using temporal decomposition. In the case of lattice Boltzmann, this means that temporal decomposition will only be employed when the number of fluid nodes per core drops below the cutoff defined by Fig. fig:dropoff. Adhering to this drop off imposes a fundamental limit to the potential number of fluid nodes to be handles on each core within a $T2$ group and subsequently a limit to the potential message size being sent between K iterations.

7.8 Theoretical Parallel Speedup

In order to gain insight into the performance that can be expected from the use of this technique, an upper bound on the strong scaling capabilities are calculated.

To this end, it is assumed that the hardware system is homogenous in that each processor is identical and that the communication time between the processors is negligible. The time for one processor to compute one time step by the fine iterator is denoted by τ_F and similarly, the time to compute one time step of by the coarse iterator is denoted by τ_C. The number of steps required by $\mathcal{G}$ and $\mathcal{F}$ during the course of one iteration are defined by Q_G and Q_F, respectively, and the total computational cost as γ_G and γ_F. Through use of the pipeline method shown in Fig. 7.3, each processor proceeds as soon as the initial conditions are available, and so the cost of each K iteration is given by $Q_F\tau_F + Q_G\tau_G = \gamma_F + \gamma_G$. The procedure outlined by Minion *et. al.* [102] is followed to calculate the parallel speedup, S, for pipelined parareal implementations, which is given by

$$S = \frac{N\gamma_F}{N\gamma_G + K(\gamma_G + \gamma_F)} = \frac{1}{\alpha + (K/N)(\alpha + 1)} \tag{7.2}$$

with N the total number of processors and K the number of parareal iterations, and further define $\alpha = \gamma_G/\gamma_F$. This model was used to demonstrate the speedup at different K iterations for both 10 and 100 processor runs in Fig. 7.3. The potential speedup is strongly influenced by the size of both α and K. To maximize S, both need to be minimized; however, reducing α requires that τ_G be reduced. The only way to reduce the cost of the coarse iterator is to lower the resolution and thereby make $\mathcal{G}$ less computationally intense. This reduction often leads to a less accurate coarse approximation and wider disparity between the two grids. As this disparity increases, the number of K iterations required to converge will consequently increase.

Thus, the speedup cannot be increased at will, but requires a careful balance between the ratio of time cost for the fine and coarse steps (α) and the number of iterations (K) required to achieve the desired convergence level.

Eq. 7.3 allows the estimation of speedup achieved by simply the temporal component of the space-time coupling. It can be taken alongside the speedup from spatial scaling to provide the predicated total speedup. Extending the model itself to calculate the combined speedup, E_F and E_G are defined as the parallel efficiency of the spatial decomposition on P cores. In this instance, the total number of cores in the system is $N * P$ where each N time interval is divided into P spatial domains. To calculate the total speedup use the following:

$$S_{ST} = \frac{E_F N \gamma_F}{E_G N \gamma_G + K(E_G \gamma_G + E_F \gamma_F)} = \frac{1}{\alpha + (K/N)(\alpha + 1)} \tag{7.3}$$

updating the definition of α to $(E_G\gamma_G)/(E_F\gamma_F)$. In both cases, the values of *alpha* and K strongly influence the potential speedup.

7.9 Numerical Results

The limiting factor in previous computational models has been the total extent of real time that can be simulated on current hardware. To gain a better understanding of the role parallelization-in-time plays on scalability and runtime, a series of tests were undertaken to study both the implications on the accuracy of the simulation as well as on the parallel efficiency for both a model system consisting of laminar flow

through a cylinder and flow through a patient specific arterial geometry.

7.9.1 Model Problem: Laminar Flow in a Cylinder

To study the accuracy of the results at different K iteration levels, the fluid flowing through a straight cylindrical tube of 1 cm in length and 1 mm in diameter was modeled. The flow was subject to a constant velocity at the inlet of the tube and fixed pressure gradient at the outlet. The simulation was initialized with a 100 μm resolution for the coarse iterator and a 50 μm resolution for the fine iterator. For the fine iterator, this corresponds to a cylinder with a height of 100 lattice points and a diameter of 10 lattice points. A bounding box of dimensions 10×10×100 lattice points is used to encapsulate the fine grid. This corresponds to 1200 time steps for the coarse iterator and 4800 time steps for the fine iterator, for the simulation of 0.5 seconds of flow.

in Fig. 7.6 the velocity profile is shown on a cross-section of the system at $x = 5$ and $z = 50$. The units in this case define the lattice point coordinates of the cross section. The inset of Fig. 3 shows the tube with the white line indicating the cross section used for the velocity plot. The velocity of the fluid is calculated from $(1/\rho)\sum_i f_i(\vec{x}, t)\vec{c}_i$. The deep purple line shows the result of the simulation at $K = 10$ which is the correct result of the fine iterator as this simulation used $N = 10$ temporal domains simulated on ten processors. The black line shows the velocity estimate at $K = 1$ and highlights the disparity between the initial estimate and the correct result.

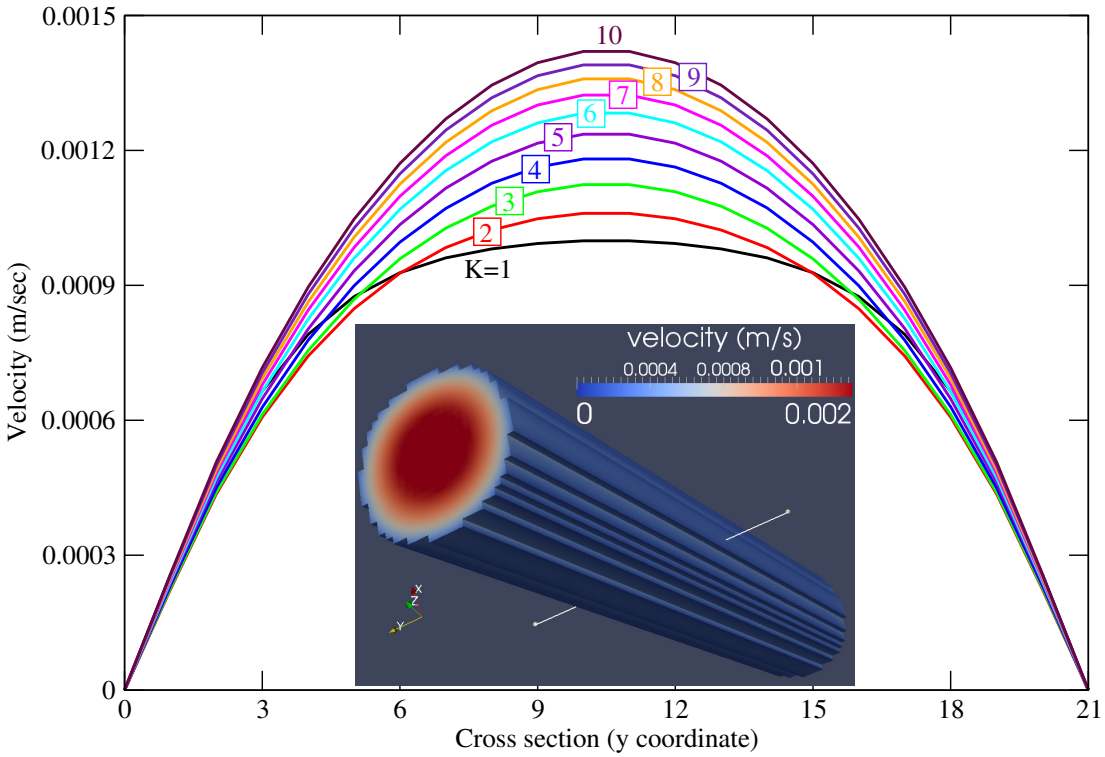
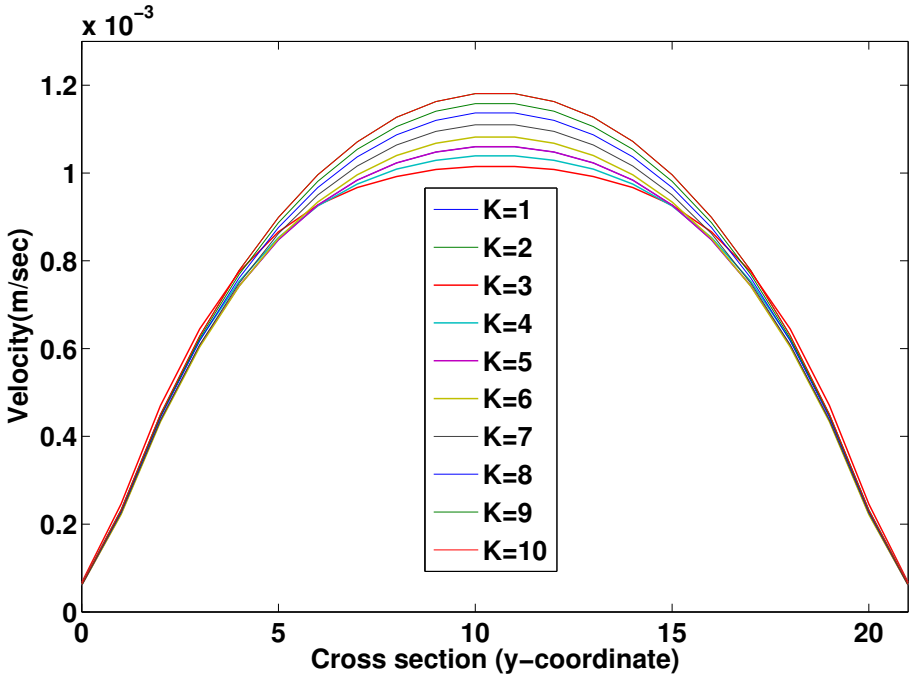
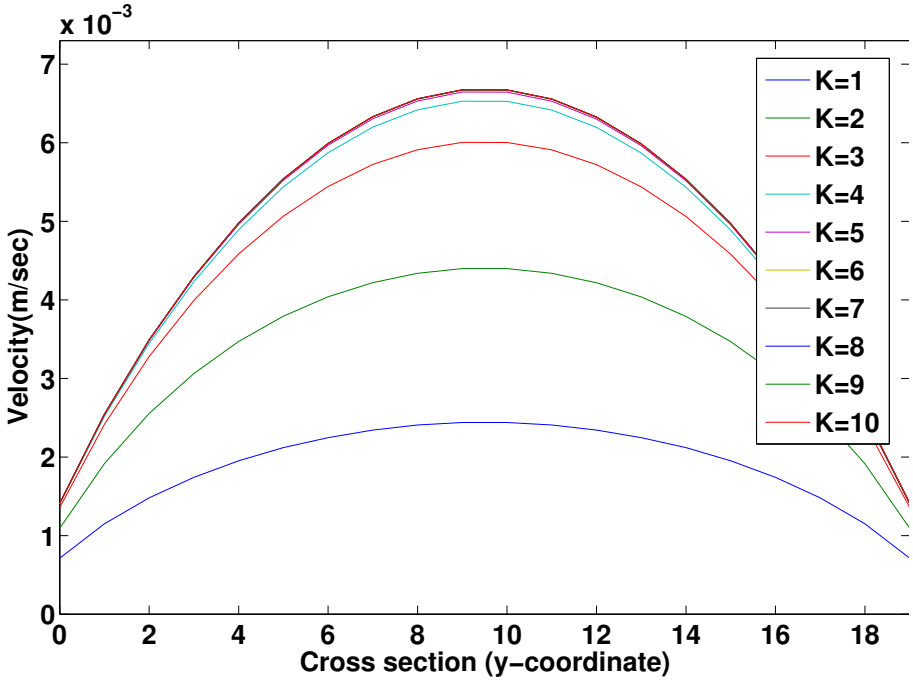


Figure 7.6: The magnitude of the velocity across the y-axis for a system broken into $N = 10$ temporal domains simulated on ten processors. The lines are labeled and color-coded according to the K value, running from $K = 1$ to $K = 10$ (converged). The inset shows the 3-dimensional velocity profile of the tube with the white line indicating the cross-section along which the velocities are plotted.



(a) Periodic Boundary Conditions



(b) Zou-He Boundary Conditions

Figure 7.7: Boundary Condition Analysis. The velocity profile of flow in the cylindrical tube at varying K levels for a simulation broken into 10 temporal domains. (a) Shows the result using periodic boundary conditions. (b) Demonstrates the use of Zou-He boundary conditions [173].

First two different boundary treatments are studied. With periodic boundary conditions, the result, shown in 7.7a, is iteratively refined in each K level moving nearer and nearer to the *correct* answer at a constant rate. Flow is imposed through the introduction of a gravitational body force. In contrast, the Zou-He boundary conditions cause more of a jump between accuracy levels leading to a convergence after only four K iterations. In Fig. 7.7b, Zou-He boundary conditions with a prescribed inflow velocity of 0.14 mm/sec and constant pressure gradient at the outlet were used. As the incorrect inlet and outlet conditions are no longer being propagated via the periodic boundary, the set inflow rate allows HARVEY to converge to the result of the fine iterator much faster.

Fig. 7.15 shows the change in accuracy across the slice at the center of a cylinder with laminar flow. The relative error is shown for the section of the cylinder shown on the left. The relative error is calculated with respect to the result of the fine solver. The reduction in error over time is clearly depicted. In this case, steady flow is used. At each K iteration, the overall accuracy increases. As you can see, the greatest error is at the center of the tube. The calculated velocity at the wall of the tube converges to the result of the fine iterator at a faster rate. This is demonstrated in Fig. 7.9 in which the relative error is shown at a range of K values at the wall and at the center of the cylinder.

Fig. 7.10 demonstrates the reduction in the time to solution at each K level. The corresponding relative error for both fluid at the center and wall of the cylinder are

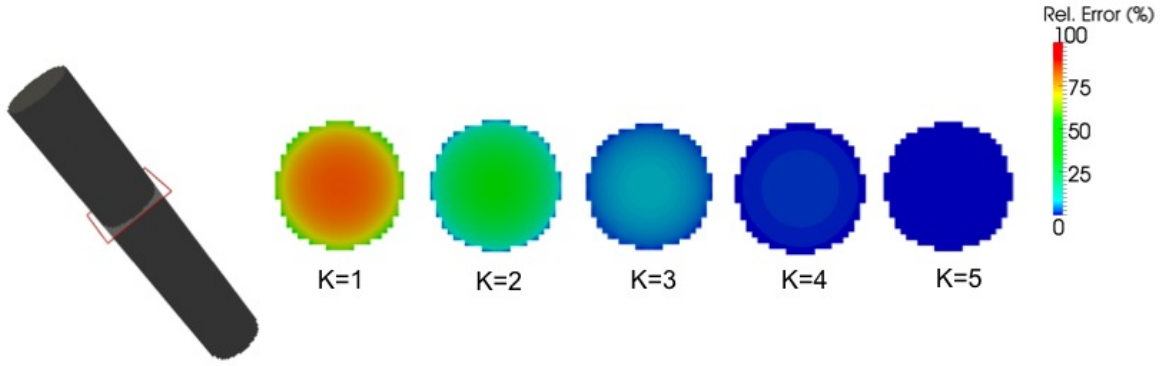


Figure 7.8: Accuracy at different K levels. The relative error is shown for the section of the cylinder shown on the left. The relative error is calculated with respect to the result of the fine solver, $\mathcal{F}$. This is for laminar flow in the cylinder and demonstrates the convergence to the fine solution as K increases. Moreover, the error variation across the section is demonstrated.

shown for each level. The dashed horizontal line depicts the wall clock time of a serial run of the fine iterator. As shown, for all $K < 9$, the time to solution is less than the serial run time. In the case of $K = 4$, the wall clock runtime was reduced by more than 50% while still having an accuracy of between 1-2% depending on whether the fluid node is at the center or wall. Fig. 4.6 further emphasizes this point by showing the corresponding parallel speedup compared to each error level. The speedup was calculated in comparison to the serial runtime of the fine iterator. If the goal of the simulation is to focus on flow at the wall, one could achieve a 3× speedup and only incur 4% error by setting $K = 3$.

Full convergence with machine accuracy to the $\mathcal{F}$ solution requires $K = N$ iterations when using N processors. The results presented and discussed in this section are intended to show that convergence within a set tolerance can be achieved with

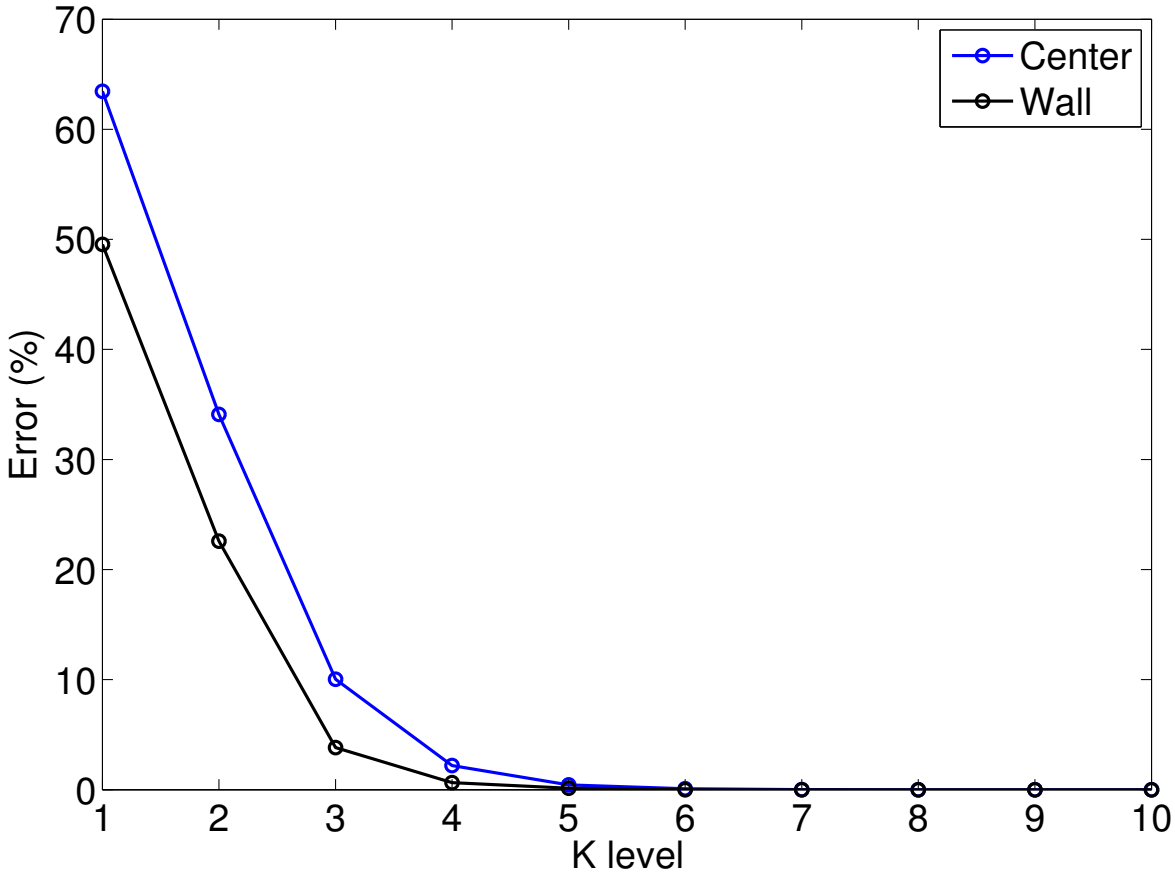


Figure 7.9: Percent relative error of the magnitude of the velocity at the wall and at the center as compared to the result of the fine solver.

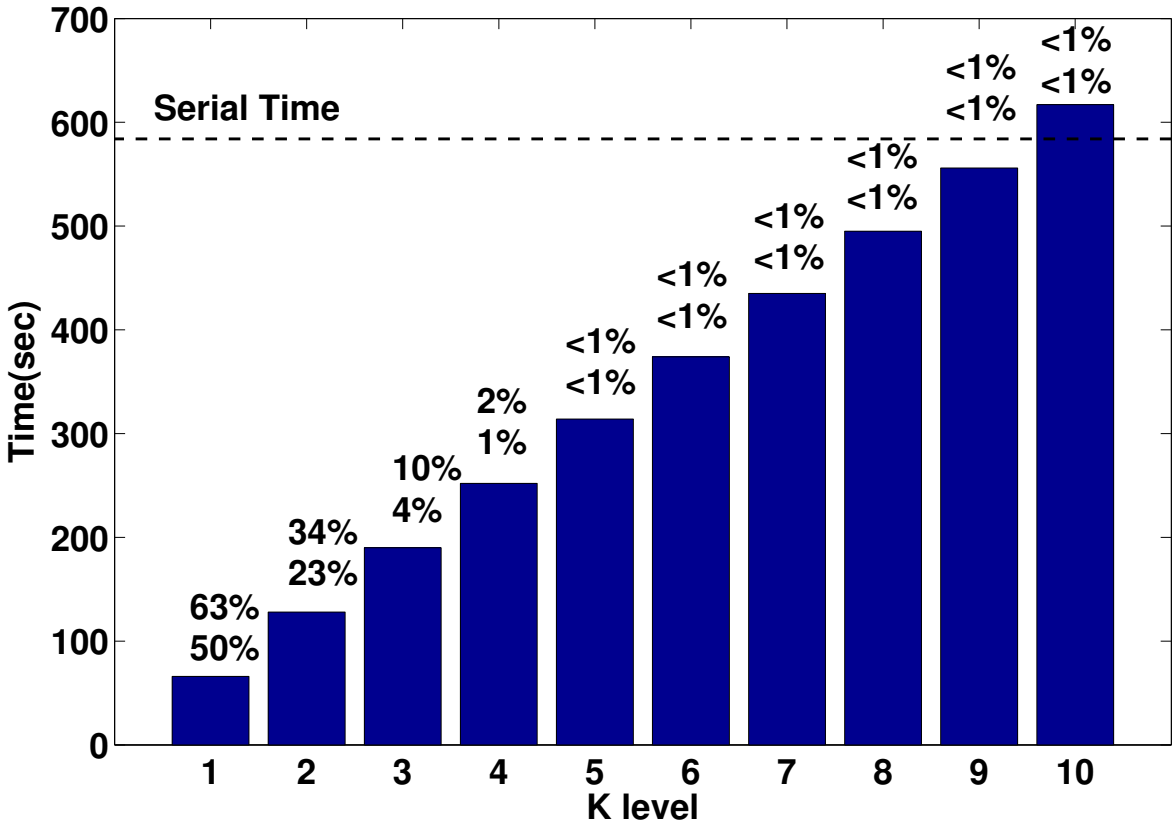


Figure 7.10: Speedup from only the temporal portion. The relative error for the flow at the center of the tube and at the wall are shown respectively above each bar.

fewer K iterations than the number of processors ($K < N$). In the initial study, ten 2.66 GHz Intel Xeon X5650 cores were used. To assess the impact of the parareal algorithm, the value of the velocity at the center of the cylindrical tube, corresponding to lattice point (5,5,50), at each K iteration was compared. This point was selected as it represents the magnitude of the velocity at the center of the tube, which is the peak of the parabolic cross-section profile. The error was assessed through comparison with the result of one serial run of the fine iterator, $\mathcal{F}$. The results are shown in Fig. 7.11 (a).

When simulating the fluid flow using simply the coarse iterator, the results have an error of nearly 30%, and with six iterations of the parareal predictor-corrector method, this error is reduced by two-thirds to less than 10%. Fig. 7.11b shows the impact of each K-level on the overall wall-clock time using a temporal break up of only 10 domains ($N = 10$). The results show a linear increase in overall runtime as more K iterations are included in the simulation, indicating that the ability to restrict to few K iterations while remaining in an acceptable convergence tolerance can lead to a large improvement in time-to-solution. For example, assuming that an error of 10% is acceptable, which corresponds to $K = 6$, gives a time to solution of 140 s, about a 30% gain compared to the time to calculate the fine iterator serially which was 197 seconds. The serial runtime is shown as the horizontal dashed line indicating that the overall time to solution for all $K < 9$ was shorter than the serial runtime.

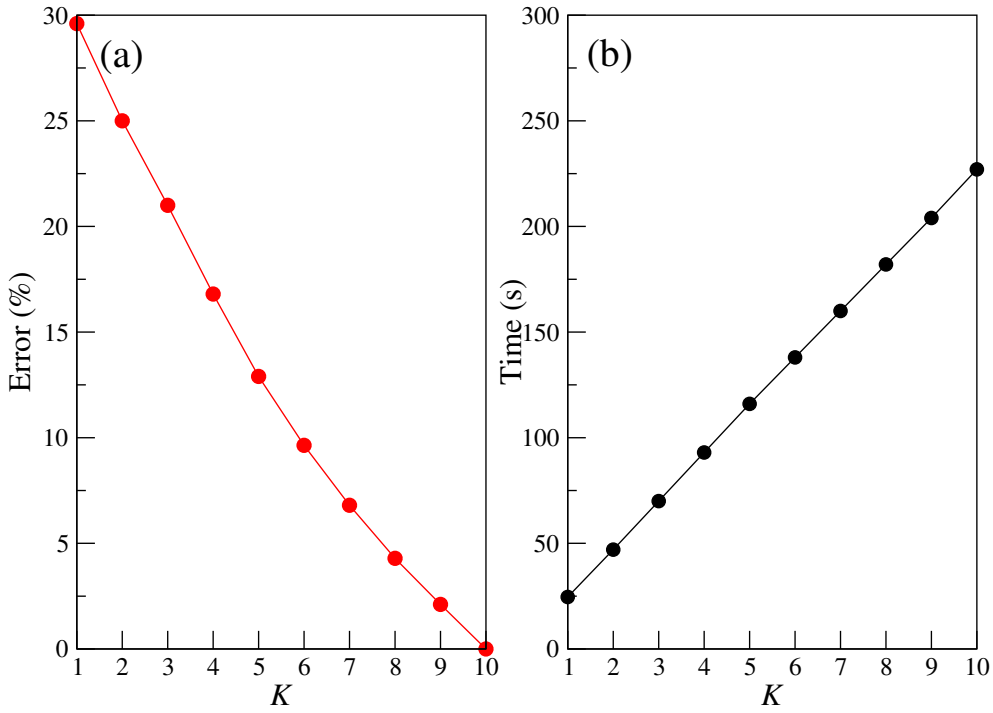


Figure 7.11: Accuracy test for a system broken into $N = 10$ temporal domains simulated on ten processors. (a) The red points show the percent error of the y-component of the velocity at point (5,5,50). (b) The wall-clock time for each K level in a ten processor run. The dashed line indicates the serial runtime.

In order to determine the success of recovering time dependent phenomena in the simulation, pulsatile flow through the cylinder was modeled. A varying inflow velocity was introduced via the Zou-He boundary conditions. In this instance, the inlet nodes are provides a set velocity and a constant pressure gradient is asserted at the outlet nodes [173]. The inbound velocity was set to oscillate on a sine wave between .011 m/s and .014 m/s. Fig. 7.12 shows the magnitude of the fluid velocity at point (5,5,50) for a range of K levels. The regions of time handled by each processor are indicated by the vertical dashed lines.

To assess the performance of this method on large systems, the initial studies were supplemented with a test using an IBM Blue Gene/P system with 2048 processors. This allowed us to assess how close the proposed method comes to meeting the theoretical performance prescribed by Eq. (7.3). Furthermore, these studies identified the peak potential speedup that could be achieved for given K iteration levels. The goal of this work was to shorten the overall time-to-solution, so the focus is shifted to the strong scaling capabilities in which a fixed system size is used as the number of cores is increased.

Fig. 7.13 shows the correlation between the theoretical performance model previously discussed for speedup, Eq. (7.3), and the experimental results of the simulation. The data shows that for given number of K-iterations, the potential speedup S for varying processor counts can be accurately estimated. The calculations are based on the coarse and fine iterators defined previously ($\Delta x_f = 50$ μm and $\Delta x_c = 100$ μm

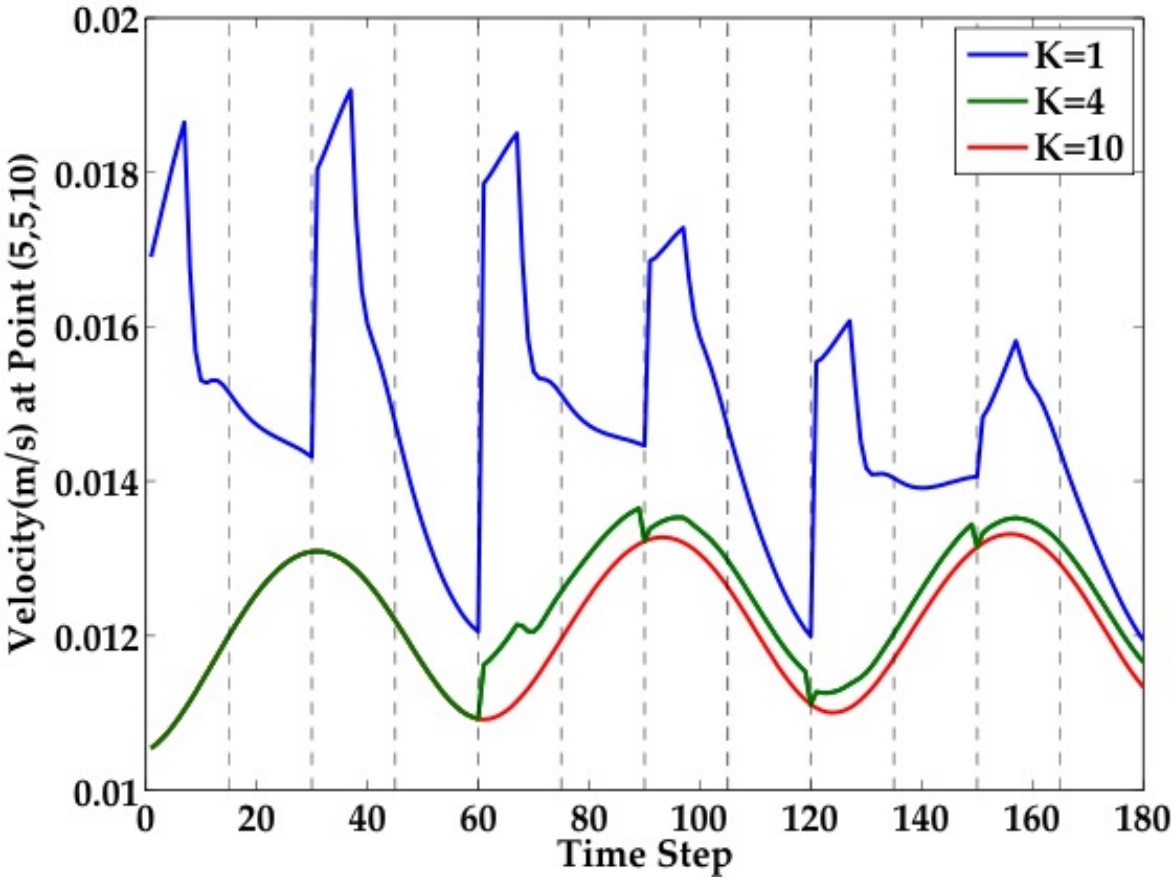


Figure 7.12: Test to recover time dependent phenomena for a system broken into $N = 10$ temporal domains simulated on ten processors. The blue line shows the magnitude of the velocity over time at point (5,5,50) after the first K iteration. The green and red lines represent $K = 4$ and $K = 10$ respectively. The vertical dashed lines indicate the break point between regions of time handled by each processor.

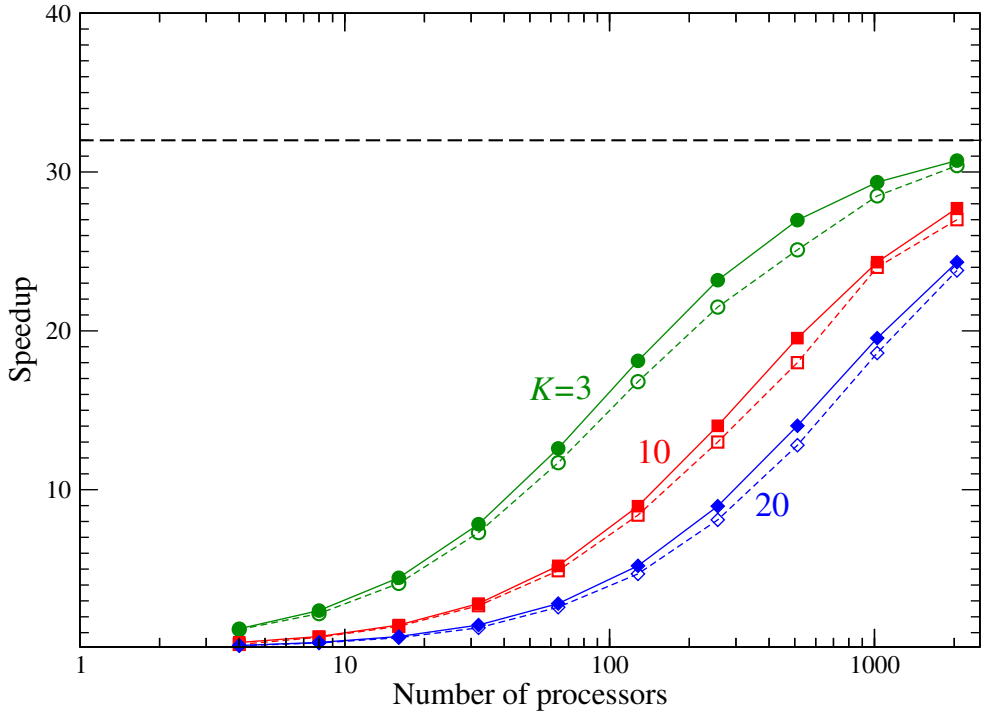


Figure 7.13: Performance tests demonstrating the strong correlation between the theoretically expected performance and experimental results. The solid lines indicate the theoretical speedup from Eq. (7.3) and the dashed lines depict the simulation results on the IBM Blue Gene/P system with 2048 processors.

resolution). These resolutions determine the value of the ratio $\alpha = \gamma_G/\gamma_F$ as they dictate the computational cost of both $\mathcal{G}$ and $\mathcal{F}$. They were chosen to ensure the stability of the simulation. A change to either grid size would inherently impact α and the associated speedup S to be gained.

Since the theoretical model of Eq. (7.3) reproduces the data very well, the limit on the potential speedup to be gained through the use of the method was also investigated. For every K iteration level, the potential speedup reaches a plateau at an upper bound value.

As Eq. (7.3) demonstrates, the potential speedup is limited by $\frac{1}{\alpha}$. With the iterators prescribed in this study, the coarse iterator is defined with a grid resolution of one half the resolution of the grid used by the fine iterator. The number of grid points in the coarse iterator is defined as $num_C = \frac{lx*ly*lz}{dx_c^3}$ and subsequently the number of time steps required is found through the following:

$$ts_c = \frac{T}{dt_c} = \frac{T}{Cdx_c^2}. \tag{7.4}$$

From these two quantities, the cost of the coarse solver is prescribed as

$$\gamma_G = \tau * ts_c * num_C = \tau * \frac{lx * ly * lz}{dx_c^3} * \frac{T}{Cdx_c^2} = \frac{\tau T lx ly lz}{Cdx_c^5} \tag{7.5}$$

Similarly, to find γ_F, in the work described here, $dx_F = 2dx_c$. This causes the number of grid points for the fine solver to be derived by:

$$num_F = \frac{lx * ly * lz}{dx_f^3} = \frac{lxlylz}{(2dx_c)^3} = \frac{lxlylz}{8dx_c^3} \tag{7.6}$$

and the time step size to be defined as $dt_f = (2dx_c)^2 = 4dx_c^2$. The number of fine time steps is then $ts_f = \frac{T}{dt_f} = \frac{T}{4Cdx_c^2}$ and the cost of the fine solver is determined through $\gamma_F = \frac{\tau Tlxlylz}{32Cdx_c^5}$. This variance corresponds to the fine iterator having four times as many time steps as the coarse iterator. Taking account the factor of 2 difference in all spatial dimensions combined with the factor of 4 difference in the number of time steps, the limit to the cost ratio, α, of the coarse to fine iterator is 32, as shown through the following:

$$\alpha = \frac{\tau Tlxlylz}{Cdx_c^5} * \frac{32Cdx_c^5}{\tau Tlxlylz} = \frac{1}{32} \tag{7.7}$$

Fig. 7.13 shows both the experimental and theoretical data confirming a plateau at a 32× speedup. The specific K for each simulation will depend on the chosen convergence tolerance, ε, which can vary in each problem. In all cases, the potential speedup reaches a limit, and beyond this point, the parallel efficiency will begin to drop dramatically.

7.9.2 Flow through Patient Specific Aorta Geometry

In this section, results of experiments on the accuracy and speedup results from space-time parallelization of the LBM as applied to understanding blood flow properties in a patient suffering from co-arctation of the aorta (CoA) are presented. Person-

alized computer simulations can provide an insightful study of the flow under stress conditions that would otherwise require difficult stress tests that have potential side effects. In the following studies, patient data from an 8-year old female with moderate aortic co-arctation (65% area reduction) is used. Gadolinium-enhanced MR angiography was performed using a 1.5-T GE Sigma scanner to obtain the arterial geometry as shown in Fig 6.1 (a). Rigid walls are assumed as well as Newtonian flow behavior for the blood, with a density r = 0.001 gr/mm3 and a dynamic viscosity m = 0.004 gr/mm/sec [81]. All of these studies were completed using an IBM Blue Gene/P supercomputer.

The simulation was initialized with a 100 μm resolution for the coarse iterator and a 50 μm resolution for the fine iterator. The fine grid corresponds to the fluid system size matching the small size in Fig. 7.1 allowing us to focus on reducing the time to solution for simulations in which adding more cores to a spatial parallelization will no longer improve the parallel performance. Unless otherwise noted, the following simulations were conducted on 32,768 cores of the IBM Blue Gene/P supercomputer. $N = 8$ was selected as the number of temporal domains so that we would be maximizing the strong scaling potential for this fluid system. By using 8 time intervals, each subcommunicator consists of 4,096 cores. As shown in Fig. 7.2, the small system achieves 85% parallel efficiency for 4,096 cores. The time duration simulated was 0.7 seconds or the average length of one human heartbeat. The goal of this work was to shorten the overall time-to-solution, so the focus here is on the strong

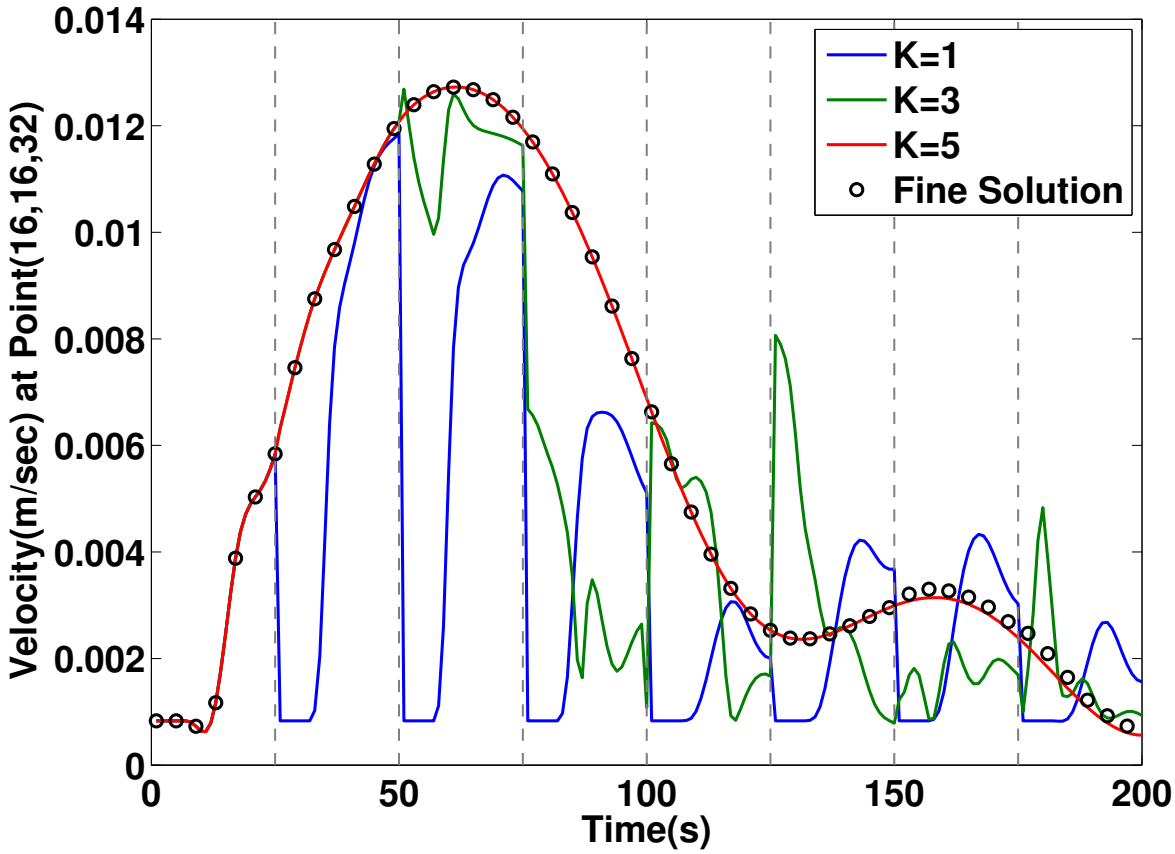


Figure 7.14: Pulsatile Flow. Test to recover time dependent phenomena for a system broken into $N = 8$ temporal domains simulated on 32,768 cores. The blue line shows the magnitude of the velocity over time at point (16,16,32) after the first K iteration. The green line, black dots, and red line represent $K = 3$, $K = 5$ and $K = 8$ respectively. The vertical dashed lines indicate the break point between regions of time handled by each core.

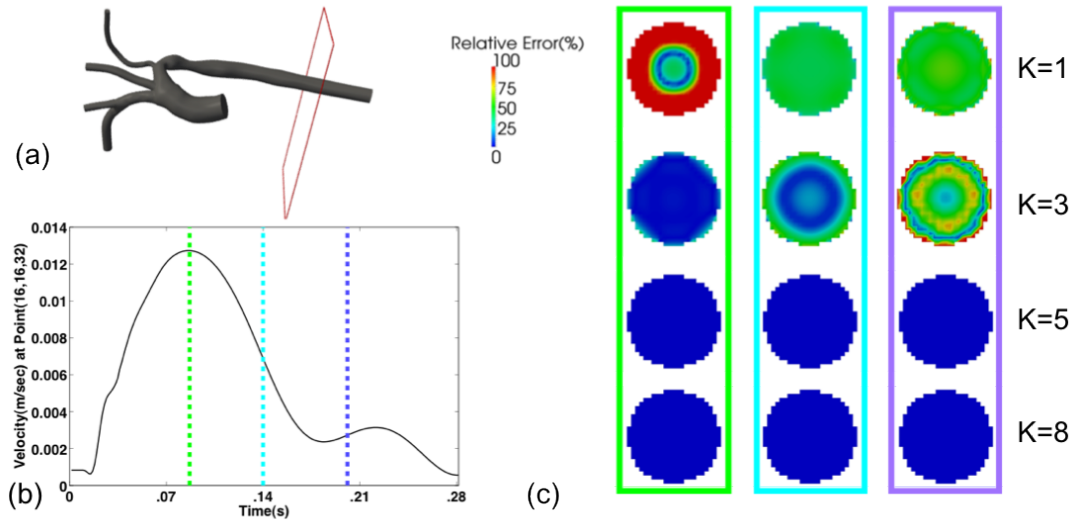


Figure 7.15: Accuracy at different K levels. (a) The mesh defining the arterial geometry from patient specific data is shown. The red rectangle depicts the section across which velocity is assessed. (b) The three vertical lines identify the time points that the error tests were imposed over the coarse of one heartbeat. (c) The relative error in velocity as compared to the solution of the fine iterator, $\mathcal{F}$, is shown at four different K levels at each time point identified in (b). The error variation across the section is highlighted.

scaling capabilities in which a fixed system size is used as the number of processors is increased.

A patient-specific inflow velocity was prescribed at the inlet and a constant pressure gradient was applied out the outlets through Zou-He boundary conditions [173]. The inflow velocity was obtained via a 2D, phase-contrast (PC) MRI sequence with through-plane velocity encoding [81]. In order to enable continuous flow throughout the heartbeat, a sum of sine functions is used to fit the data to determine the equation of the pulse. The fit procedure was performed in MatLab using a non-linear least squares method and a trust region algorithm. Using a Pearson correlation, there was

a statistically significant agreement between the phase contrast data and the equation derived velocity values ($R = 0.981, p < 7x10^{-15}$). Fig. 7.14 shows that the space-time framework in HARVEY starts to recover the pulsatile behavior with greater accuracy at each K iteration. It shows the magnitude of the fluid velocity at point (16,16,32) for a range of K levels of a simulation with 8 temporal domain slices using 32,768 cores. As the K iteration level increases, the result gets nearer and nearer to the solution of the full fine solver which is equivalent to the $K = 8$ depicted by the red line. Even at $K = 5$, the time dependent behavior is fully recovered. The dashed black vertical lines indicate the break point between regions of time handled by different time intervals or subcommunicators.

Full convergence with machine accuracy to the $\mathcal{F}$ solution requires $K = N$ iterations when using N processors. The results presented and discussed in this section are intended to show that convergence within a set tolerance can be achieved with fewer K iterations than the number of processors ($K < N$). Fig. 7.15 shows the change in accuracy across the slice at the red plane within a real patient's arterial geometry as shown in Fig. 7.15(a). The relative error in velocity as compared to the solution of the fine iterator, $\mathcal{F}$, is shown at three different time points from within a heartbeat. The pulsatile nature of the flow causes a variation in the error. At each K iteration, the overall accuracy increases. When $K > 5$, the relative error across the entire slice is approximately zero. The fact that the initial time intervals reach convergence first is highlighted by $K = 3$ in which the error is greatly reduced for

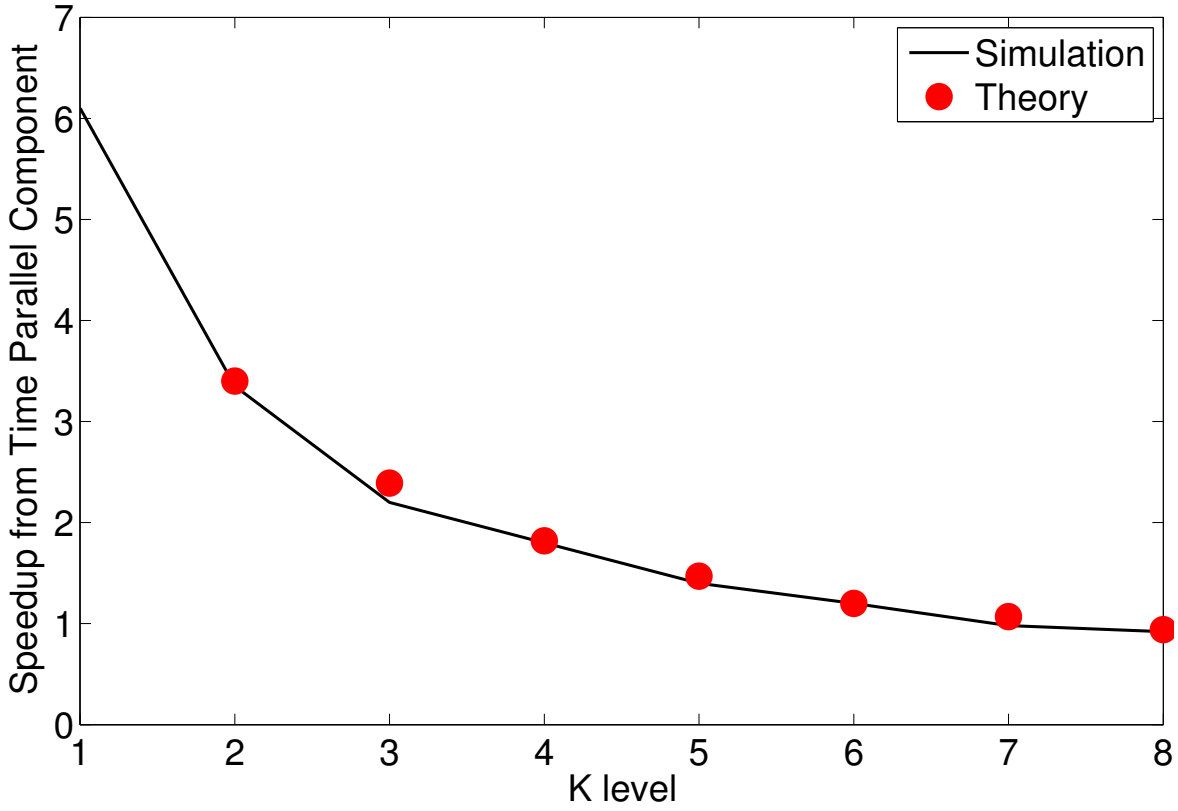


Figure 7.16: Performance tests demonstrating the strong correlation between the theoretically expected performance and experimental results. The black line depicts the simulation results and the red circles indicate the theoretical speedup added from the temporal component as calculated from Eq. (7.3).

the first two time points. Of further note, the greatest error is at the center of the tube. The calculated velocity at the wall of the tube converges to the result of the fine iterator at a faster rate. This is significant in selecting the desired K level as it can depend on the research question and disease targeted. Often when assessing risk for a disease like atherosclerosis, one is concerned with the magnitude of the endothelial shear stress on the wall of the vessel [100]. In this case a lower K iteration may provide the accuracy desired. Conversely, when trying to understand the pressure gradient associated with co-arctation of the aorta, the pressure at the center of the vessel is equally as significant.

These results are congruent with results found for other domains. Baffico *et al.* showed that $K = 4$ of a domain broken into six temporal intervals provided accurate results for a molecular dynamics code [9] and Fisher *et al.* demonstrated high accuracy at $K = 2$ for a Navier-Stokes simulation with ten temporal intervals [50].

The additional speedup provided by this method above and beyond that achieved by the spatial parallelization was then assessed and the disparity between this implementation and the theoretical performance prescribed by Eq. (7.3) evaluated.

Fig. 7.16 shows the correlation between the theoretical performance model previously discussed for speedup, Eq. (7.3), and the experimental results of the simulation. The previously mentioned resolutions ($\Delta x_f = 50$ μm and $\Delta x_c = 100$ μm resolution) were used to determine the value of α and consequently the overall computational costs. Any change to either grid resolution would impact the associated speedup that

can be achieved.

Fig. 7.17 demonstrates the reduction in the time to solution at each K level. The corresponding relative error for both fluid at the center and wall of the cylinder are shown for each level. The bold red horizontal line shows the wall clock time of parallel run on the full 32,768 processors with only spatial parallelization used. The significance of this data is that available hardware can be utilized more efficiently by combining the use of temporal and spatial scaling. As the data shows, for all iterations with $K < 9$, the time to solution is less than the spatial only run time. In the case of $K = 4$, the wall clock runtime was reduced by more than 50% while still having an accuracy of between 2-5% depending on whether the fluid node is at the center or wall.

7.10 Discussion

For many fluid problems even beyond the medical applications discusses in this paper, there is a strong need to model longer time durations for fixed system sizes. In these instances, there is often a fundamental limit to the benefits that can be obtained through traditional spatial scaling. Through the careful coupling of temporal with spatial parallelization, it has been demonstrated that this method provides an efficient way to leverage available parallel resources and to reduce the time to solution for real problems and real data.

In this work, a time-parallel method to speedup fluid simulations based on the lat-

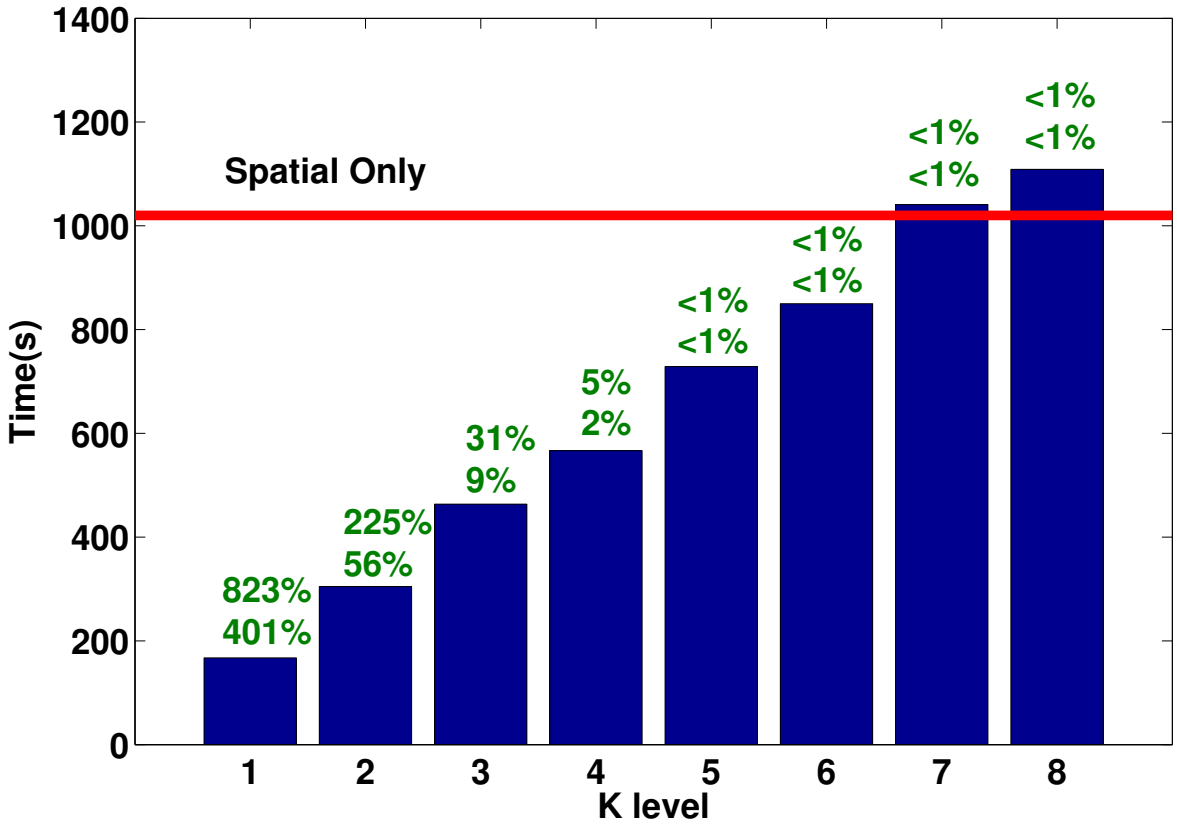


Figure 7.17: Time to solution for each K level as compared to the serial run. The relative errors for the flow at the center of the tube and at the wall are shown respectively above each bar. The dashed horizontal line represents the serial runtime.

tice Boltzmann method was first introduced, thus enabling the modeling of more time steps in a shorter wall-clock time. Previously, a limiting factor to parallel efficiency of simulations of a fixed problem size was the saturation of spatial parallelization. While LBM is well suited to large-scale spatial parallelism, parallel efficiency starts to decline as the number of mesh points per processor becomes too low. Typically, in these simulations there is a fixed resolution requirement in which further discretization of the space will not yield better accuracy. In such cases, when the limits of spatial parallelism have been reached, the method introduced here to parallelize the LBM in time overcomes this intrinsic strong scaling limit.

In order to use a time parallel LBM code on large scales efficiently, carefully designed coarse and fine iterators are needed. It was demonstrated that this goal can be met by means of the multigrid approach. By combining mesh refinement methods with the parareal algorithm, longer time intervals of a fluid simulation were able to be simulated in a shorter wall-clock time, within which, through iterative refinement, the compute-intensive fine iterator is modeled with temporal parallelization. Thus, the novel combination of mesh refinement and the parareal algorithm presented in this work provide a method to considerably extend the intrinsic strong scaling limit of classical LBM codes and therefore minimize the runtime of fixed-size fluid simulations.

Finally, here are some comments on some issues related to future extensions and further development of the method. The implementation described here requires that processors that have achieved convergence with the fine iterator remain idle until the

solution for the entire time domain has converged within the desired tolerance. The parallel efficiency of this method can be improved through reuse of these processors. One such option would be to re-allocate their use for further spatial discretization of remaining time intervals. Another subtle point is that if a grid point is defined as a boundary node on the coarse mesh, it will translate to a wall plus fluid nodes in the fine mesh. The value of the fluid quantities near the wall will be inaccurate as they are initialized to zero due to the averaging from the coarseness. The additional fluid nodes encompassed by the fine grid have contributions from coarse wall nodes in their initialization. The contribution from the wall nodes lowers the real distribution value seen by the fluid at the wall. Through each K-iteration, the value will be refined and propagated to the mesh. After the first iteration they will be based on values from the first fine full iteration and propagated through the system. This may take more time but only impacts the few lattice points directly next to the wall in the fine grid. In an extremely porous material, this could be an issue but otherwise (as in the case of the model system presented here) it does not have serious adverse effects. For porous materials, an interpolation function should take nearby fluid data to initialize the conditions instead of using the zero contribution from the wall.

8

Accounting for Deformational Forces

Nothing happens until something moves.

– Albert Einstein [131]

8.1 Motivation

While it is accepted that complex flow fields in coronary arteries are related to the development and progression of CVD, most numeral studies rely on geometry, pulsatile flow, and the non-Newtonian behavior of blood in non-moving vessels [120]. The coronary arteries are located such that they curve around the myocardium, the muscular tissue of the heart, and throughout the cardiac cycle, the coronary arteries undergo large dynamic variations in curvature due to this position on the beating

heart. In 1976, Gibson *et al.* studied the left ventricular angiograms of 60 patients with heart disease and 10 normal patients frame by frame to evaluate the vessel motion throughout the cardiac cycle. This study demonstrated significant wall movement over the coarse of the heartbeat both for normal and diseased patients [57]. More recently, Gross and Friedman specifically evaluated the curvature variation of the left anterior descending (LAD) artery over the coarse of a heartbeat and demonstrated a significant change [65].

A useful quantification to assess the range of curvature change in the coronary arteries is referred to as the curvature ratio, δ, and defined as $\delta = \frac{a}{R}$. In this case a denotes the arterial radius and R indicates the radius of curvature. For the left coronary tree, δ typically falls in the range of 0.02-0.5, with a varying between $3 - 4mm$. Throughout the cardiac cycle, the expansion and contraction of the myocardium leads to the aforementioned curvature change and displacement of the coronary arteries. Santamarina *et al.* defined this change in curvature as ϵ or the ratio of the amplitude of the change in radius, ΔR, to the average radius of curvature shown in Eq. 8.1 [137]. Experimental measurements of the left anterior descending artery have determined ϵ to fall between 0.7 and 0.8 (c.f. [123], [65]). While the effect of curvature on flow in tubes has been a subject of heavy investigation (summarized well in [110]), accounting for the dynamic curvature change in time that poses new challenges.

$$\epsilon = \frac{\Delta R}{R_{mean}} \tag{8.1}$$

Initial experiments have shown the importance of accounting for flow pulsatility when modeling hemodynamics in coronary arteries by measuring steady and pulsatile flow through vascular casts (c.f. [135], [91]); however, it wasn't until recently that studies have been undertaken to gain insight into the impact of the cardiac-induced motion on coronary flow. In 1998, Santamarina *et al.* conducted an experiment in which a uniformly curved tube was fixed in place at its inlet and had the radius of curvature varied in time in order to assess flow patterns of a simplified geometry similarly shaped to coronary arteries. Flow patterns and wall shear rates were determined for steady inflow while curvature was varied sinusoidally in time at a frequency of $1Hz$. When compared to results from flow through curved tubes fixed with the static radii of curvature equal to the minimum, mean, and maximum exhibited in the oscillating experiment, it was shown that the dynamic deformation had significant impact. The wall shear rates vary as much as 52% of the static mean wall shear rate when dynamic deformation was included [137]. These results proved that it is not simply enough to account for the change in curvature, but rather the smooth transition between curvature states has to be accounted for.

This effect was further shown in more simplistic models of the right coronary artery (RCA) ([103], [125]) as well as in simple patient geometries like a symmetric bifurcation [164] and a non-compliant model of the LAD and it's first diagonal branch. Torii *et al.* introduced one of the first subject-specific models of an RCA by interpolating between geometries acquired at multiple time points [157]. The dynam-

ically varying vascular geometry was reconstructed from magnetic resonance images (MRI). The effects of the vessel motion on shear stress was examined through the comparison of an RCA model with time-varying geometry compared to those with a static geometry corresponding to 9 different time points in the cardiac cycle. A method to leverage two-dimensional cross-sectional images that are obtained via an interleaved spiral k-space technique with short acquisition windows on the order of 10 ms was developed. This technique enabled imaging at any time-point without motion blurring or the need for full three-dimensional image acquisition at multiple time points in the heartbeat [157]. Torii *et al* further studied the impact the dynamic vessel motion had specifically on the the resulting endothelial shear stress. Time-varying curvature change was introduced by interpolating geometries obtained at 14 time points during the cardiac cycle using MRI. These experiments confirmed the significant impact that dynamic curvature change could have on associated wall shear stress [156].

Pivkin *et al.* extended these studies to investigate the role that the combination of both pulsatility and curvature variation has on the wall shear rate at the bifurcation of a coronary artery. The increase in curvature during ventricular expansion, decrease during contraction, and the corresponding flow rates are taken into account. An arbitrary Lagrangian Eulerian (ALE) formulation of the unsteady, incompressible, three-dimensional Navier-Stokes equations is employed to solve for the flow field. In this work, a representative geometry was created as an analytical intersection of two

cylinders. The motion of the cardiac muscle was simulated y changing the curvature radius of the main cylindrical tube over time. Velocity smoothing is used to update the computational mesh defining the arterial geometry. The effect of pulsatile inflow alongside the time-varying geometry was shown to have a considerable impact on the flow dynamics and specifically on the wall shear rate [120].

The studies mentioned above concretely demonstrate not only the contribution that arterial curvature has on hemodynamic factors like wall shear stress, but the essential need to include its dynamic temporal change in order to obtain accurate numerical estimates of such quantities. As the changes in flow patterns have been shown to potentially be more important than the flow patterns themselves in determining the locations of potentially deleterious effects on the vascular walls [28], it is important to adequately account for the changes in flow dynamics. The aforementioned studies suffer from two common drawbacks. First, most rely on simplified geometrical representations of coronary arteries. Those that do rely on data acquired for medical imaging technologies, focus on a single artery or at most the inclusion of one simple bifurcation. The goal of this thesis is to enable efficient and accurate modeling of larger arterial sections such as the entire coronary arterial tree. This results in a much more complex geometry including at least 12 main coronary arteries. Second, in all cases the definition of the arterial geometry is modified throughout the cardiac cycle to account for the curvature changes. This is typically employed through the interpolation between static geometries obtained through multiple MRI

acquisitions. Not only can this interpolation become computationally expensive for large arterial trees, but also the need for many acquisition time points increases the patient radiation dosage. In this chapter, a method to account for the deformational forces exerted on the blood flow due to the curvature changes throughout the cardiac cycle while leaving the geometry itself static is proposed. The numerical method will be discussed followed by experiments demonstrating the capture of the effects of dynamic curvature change in a computationally efficient manner.

8.2 Definition of External Force

In order to account for the deformational forces in the LBM leveraged in work presented in this thesis, the introduction of a body force that encompasses the deformational forces is used. As mentioned, a uniform curvature at each individual time point is assumed requiring an isotropic and homogeneous system view. To incorporate the curvature changes into the model, the underlying lattice grid applied is expanded uniformly between time points to enforce that the relationship between the different grid points remain the same while the density of the overall system changes.

This approach is similar to kinetic models of the expanding universe. As described by H.P. Robertson in [132], the general theory of relativity takes the view that on sufficiently large scales, the universe on average is homogeneous and isotropic. This is also known as the *cosmological principle*. In both the de Sitter and Einstein models of cosmology, the curvature of the universe remains constant in time [14]. This achieved

through the use of *co-moving* coordinates in which the grid system applied to a galaxy is assigned values that remain constant as the universe expands. To clarify, the distance between the grids points themselves will actually increase to account for the expansion of the underlying manifold. The Robertson-Walker metric, as named for the two researchers who first derived it in cosmology, requires that the large-scale curvature is the same at each location in each time point, similar to the requirements imposed on the representation of the curvature of coronary arteries [39]. A single cosmic scale factor, $R(t)$, defines the relative expansion of the universe as a function of time. It is used to relate the distance between two objects over time.

Taking a kinetic formalism of the universe, the fundamental quantity in the description of fluid particles in the expanding universe is the distribution function, f, similar to the function described in Chapter 2 to capture hydrodynamic behavior. This quantity is modified to be a function of position, velocity, and now $R(t)$ instead of simply time. The Liouville operator which asserts that the f is constant along the trajectories of the system is thus defined by Eq. 8.2.

$$L(f)\frac{\partial f}{\partial t} - 2\frac{\dot{R}}{R}\rho\frac{\partial f}{\partial \rho} \tag{8.2}$$

The distribution function is strongly influenced by the microscopic behavior of the inter-particle collisions. It is therefore useful to recast the Liouville operator in terms of the local momentum. To do so, the method put forth by Bernstein is used alongside the assumption of massless particles resulting in Eq. 8.3. Readers are referred to [14]

for more in depth background.

$$L(f) = \frac{\partial f}{\partial t} - \frac{\dot{R}}{R} v_i \partial v f \tag{8.3}$$

The underlying manifold is expanding and increasing the distance between prescribed grid points; however, the relationship between the grid points themselves remains the same. The Bernstein method for introducing forces due to the increase of the radius of curvature of the universe is used to define the additional body force resulting from the expansion and contraction of the heart.

$$\frac{\dot{R}}{R} = \frac{-\Delta D \omega sin(\omega t)}{Ro + \Delta D cos(\omega t)} \tag{8.4}$$

In this case the variable $\frac{\dot{R}}{R}$ from Eq. 8.3 is modified as shown in Eq. 8.4 where ΔD defines the average change in diameter of the system, Ro defines the initial radius of the heart, and *omega* controls the curvature change over time. In this thesis, ΔD, the average change in diameter throughout a cardiac cycle, is set to $1cm$ [163]. The average radius of the heart is defined as $2.5cm$ and drawing from the work by Santamarina *et al.* [137] the curvature change is prescribed as a sinusoidal function, $\omega = \frac{2\pi}{0.7s}$.

8.3 Inclusion of External Force Term in the LBM

Shan and He demonstrated the equivalence between the solution of the LBM and the approximations to the Boltzmann equation by Hermite polynomial expansion as a new and alternate approach for discretizing the Boltzmann Equation in velocity space [138]. The truncation of the Hermite expansion of the Boltzmann distribution was shown to be equivalent to using the LBM for selected discrete velocity models, including the previously described D3Q19 model.

In this section, we follow the procedure laid out in [139] to define the external forcing term by casting it in the Gauss-Hermite formalism. A detailed description of the relationship between hydrodynamic moment integrations and hydrodynamic quantities like density and velocity is presented followed by the extension to include the external force term. By expanding $f_i(\vec{x} + \vec{c}_i\Delta t, t + \Delta t)$ as defined by Eq. 5.1 in terms of dimensionless Hermite ortho-normal polynomials in velocity space xi, as outlined in [139] f is recast as:

$$f(\vec{x}, \xi, t) = \omega(\xi) \sum_{n=0}^{\infty} \frac{1}{n!} a^{(n)}(x, t) \mathcal{H}^{(n)}(\xi) \tag{8.5}$$

The dimensionless expansion coefficients, $a^{(n)}(x, t)$, are defined by

$$a^{(n)}(x, t) = \int f(\vec{x}, \xi, t) \mathcal{H}^{(n)}(\xi) \partial\xi \tag{8.6}$$

Eq. 8.6 demonstrates that all expansion coefficients are linear combinations of the

velocity moments of f. Shan *et al.* demonstrated that the first few expansion coefficients are directly associated with conventional hydrodynamic variables and can be defined as follows [139]:

$$a^{(0)} = \int f \partial\xi = \rho \tag{8.7}$$

$$a^{(1)} = \int f \xi \partial\xi = \rho u \tag{8.8}$$

$$a^{(2)} = \int f(\xi^2 - \delta)\partial\xi = P + \rho(u^2 - \delta) \tag{8.9}$$

Eq. 8.7-8.9 demonstrate that the thermodynamic variables can be expressed in terms of just the first three Hermite expansion coefficients. In this case, density is defined in Eq. 8.7, momentum in Eq. 8.8, and the momentum flux tensor, P, in Eq. 8.9. Martys *et al.* built on this formalism to introduce the inclusion of a body force term [93]. In this case, the external force term, $F(\xi)$, is defined as $\overrightarrow{g} \cdot \partial v f$ and Eq. 8.10 is the Hermite expansion in which $ga^{(n-1)}$ is the symmetric tensor product of g and $a^{(n-1)}$.

$$F(\xi) = \omega \sum_{n=1}^{\infty} \frac{1}{n!} g a^{(n-1)} \mathcal{H}^{(n)} \tag{8.10}$$

As the leading coefficients are defined by Equations 8.7-8.9, the second order expansion of F can be denoted with Eq. 8.11.

$$F \cdot \partial v f = \omega(\xi)\rho \left[0 + \overrightarrow{g} \cdot \overrightarrow{\xi} + (\overrightarrow{g} \cdot \overrightarrow{\xi})(\overrightarrow{u} \cdot \overrightarrow{\xi}) - \overrightarrow{g} \cdot \overrightarrow{u}\right] \tag{8.11}$$

Following this same procedure for the external force from the curvature change as defined by $\frac{-\dot{R}}{R}\overrightarrow{v} \cdot \partial v f$, Eq. 8.12 is derived.

$$F(v) \cdot \partial v f = \frac{\dot{R}}{R}\omega\xi\rho \left[1 + 2\overrightarrow{u} \cdot \overrightarrow{u\xi} + (3\hat{P} + 2c^2\rho\hat{I}) : (\hat{\xi\xi} - c^2\hat{I})\right] \tag{8.12}$$

8.4 Numerical Results

To evaluate the accuracy of the proposed method of capturing the impact on shear stress and velocity profiles of time varying curvature change, a series of tests were undertaken on two representative geometries: a curved tube and a patient-specific left coronary tree.

8.4.1 Model Problem: Flow in a Curved Tube

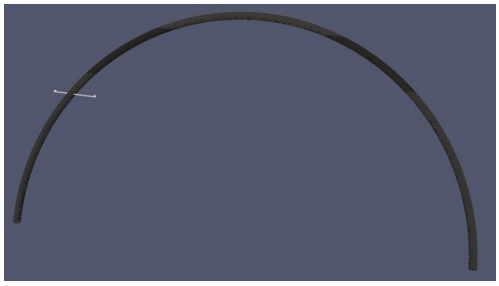
First, a computational model mimicking the *in vitro* experiment conducted by Santamarina *et al.* was constructed [137]. In this case, a curved section of a tube with a1.27cm inside diameter was surrounded by a spiral vacuum hose to prevent the distortion of the cross sectional area. The other geometric parameters were $\delta_m = 0.043$, $\epsilon = 0.26$, and Re=300. One end of the tube was held fixed while the curvature change was introduced through a motor coupled to a crank-piston linkage that introduced

the sinusoidal oscillation. The fluid in the system was a water-glycerin solution of 65% glycerin by volume and had a kinematic viscosity of $0.12cm^2/s$. These conditions were replicated in the simulation.

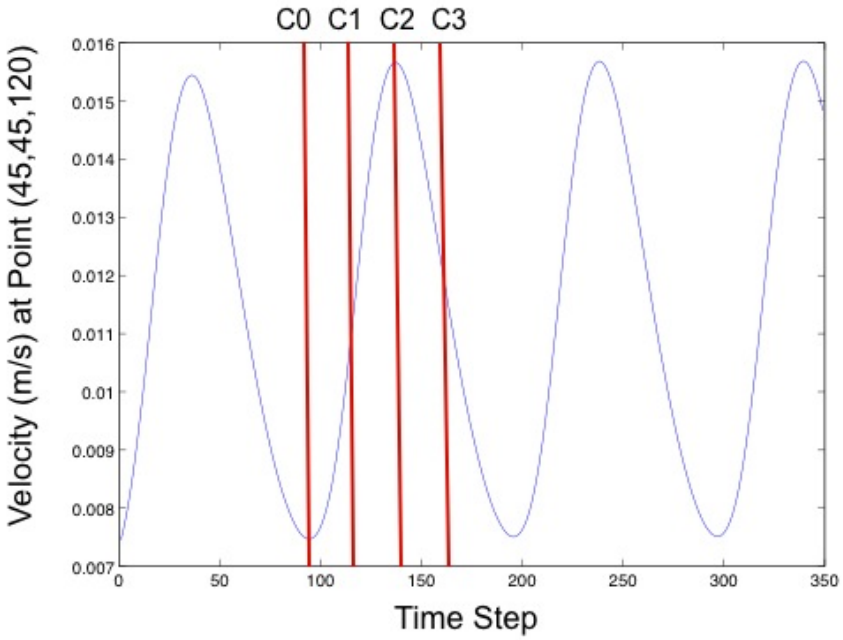
The geometry of the curved tube, shown in Fig. 8.1 (a), was similar to that of typical coronary arteries. Fig. 8.1 (b) shows the velocity magnitude at the point (45,45,120) over the coarse of 350 time steps. The impact of the curvature change has been introduced as a sine wave applied uniformly across the geometry using the introduction of the external force defined in Eq. 8.12. Over the cardiac cycle, the velocity fluctuation recovers the sinusoidal deformation wave form. Fig. 8.1 (c) shows the velocity profile across the spatial slice of the tube indicated by the white line in (a) taken at each checkpoint (C0, C1, C2, and C3) defined in (b). Again, as the cardiac cycle progresses, a significant qualitative change in flow patterns for the fours checkpoints in the figure, which reflect the waveform of the curvature change.

8.4.2 Flow through Patient Specific Coronary Arterial Tree Geometry

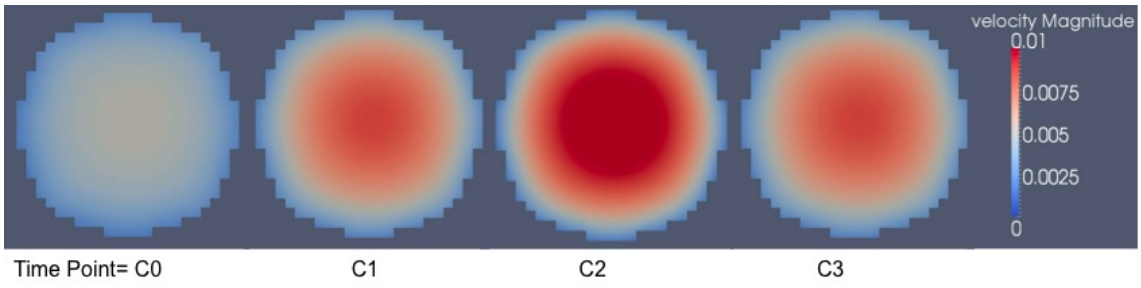
The overall impact of the deformational forces on shear stress in a real geometry was evaluated using the left coronary arterial tree obtained from multidetector computed tomography (MDCT). This geometry was acquired in a single heartbeat at $0.5mm$ resolution as described in Chapter 4. In all simulations, a steady flow was used to put emphasis on the impact from the forces resulting purely from the curva-



(a) Simplified Geometry



(b) Curvature Variation

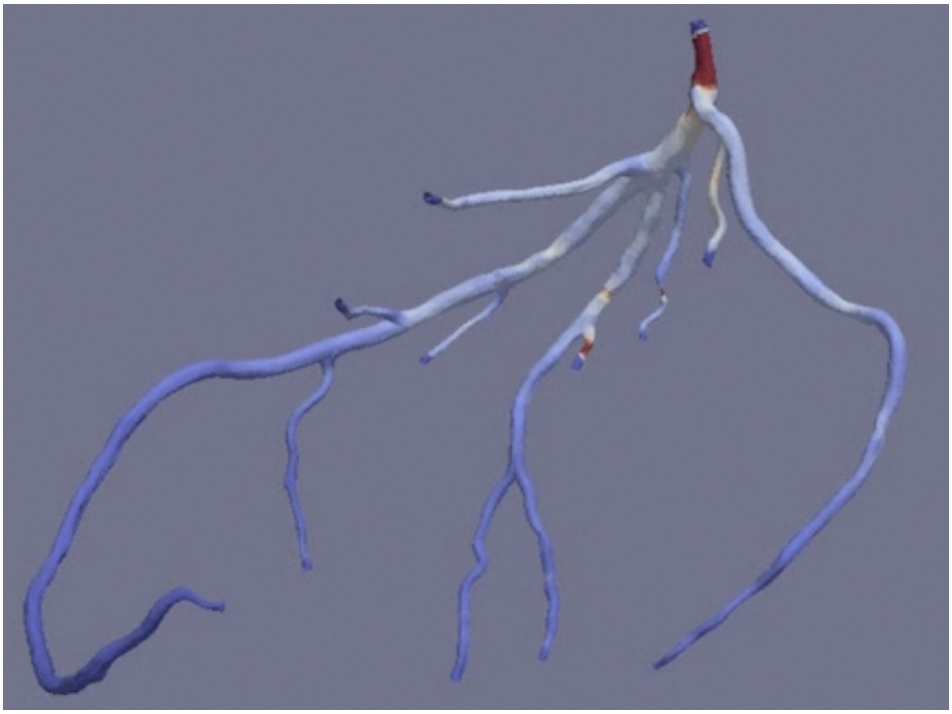


(c) Velocity Profile

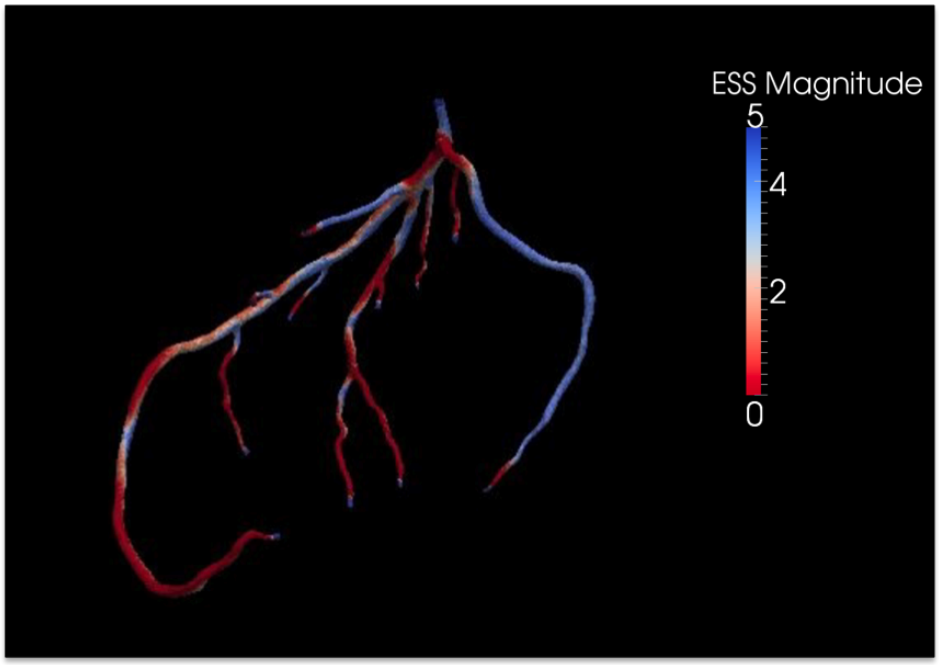
Figure 8.1: Results of flow in a simple curved tube similar to known coronary geometry. (a) The geometry with $\delta_m = 0.043$ and the average radius of curvature of $0.635cm$. (b) The magnitude of the velocity at point (45,45,120) is shown over the coarse of the cardiac cycle. (c) The slices represented here are taken from the area marked by the white line of (a) and at the temporal checkpoints designated by the vertical red lines in (b).

ture change as the heart expands and contracts. Fig. 8.4.2 (a) the conventional result using a steady flow of $0.14m/s$ is shown. This is the result after the shear stress has converged to a steady state. Fig. 8.4.2 (b-d) depict the shear stress map at three time points in the expansion cycle. In this case, the curvature change is being represented as a sinusiodal wave that is applied uniformly across the arterial tree. The sine wave was scaled to match the time duration of the average heartbeat, 0.7 seconds [163]; however, this does result in the peak of the expansion occurring at $t = 0.35s$. In future work, the sine waveform would be replaced with the patient's cardiac inflow waveform.

In order to better assess the impact of the deformational forces on the shear stress across the individual arteries, Fig. 8.3 shows a 2D projection of the LAD artery in which the width equals the circumference of the artery at the cross section. ESS is mapped to the surface using a color encoding that diverges from red to blue. In a formal quantitative user study with domain experts, this 2D representation and color map resulted in fewer diagnostic errors and faster identification of at risk regions [18]. Using this representation, one can see the impact that the curvature change has at different points in the arterial wall throughout the cardiac cycle. Again, the cardiac cycle is implemented as an idealized sinusoidal wave with the maximum point of expansion occurring at $t = 0.35s$. For the time point at $0.35s$, both the result from the standard force and the result when including the deformational forces are shown. This emphasizes the change in ESS that occurs due to the introduction of

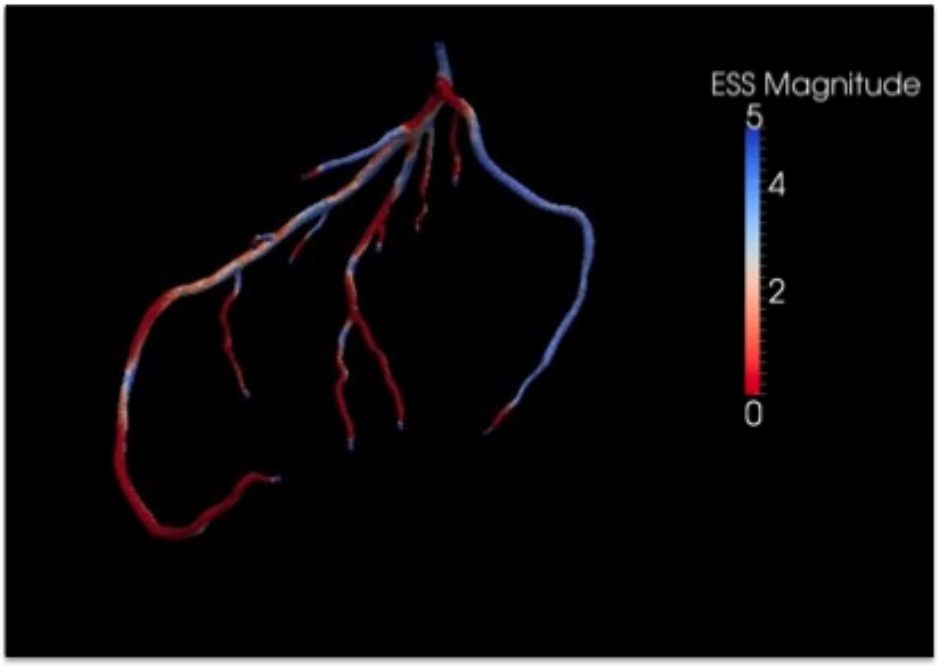


(a) Standard Force

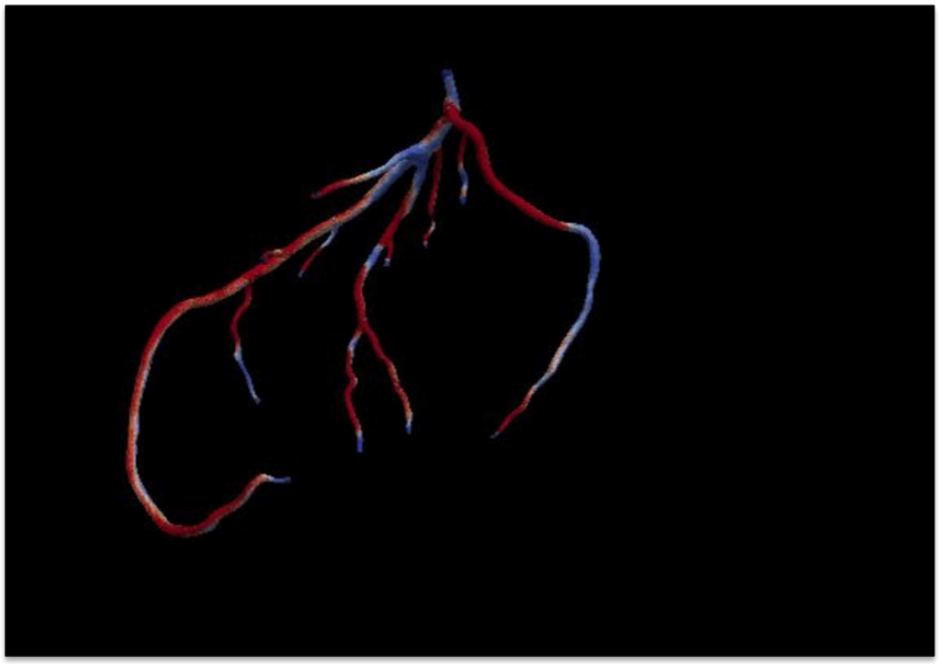


(b) t=0s

Figure 8.2: First two images of the evaluation of the impact of the deformational forces on the endothelial shear stress. (a) Depicts the shear stress of a steady flow through the patient specific geometry once it has converged to a steady state. (b) Shear stress mapping for simulation including the deformational forces at the initialized state.



(c) t=0.35s



(d) t=0.7s

Figure 8.2: (Continued) Second two images of the evaluation of the impact of the deformational forces on the endothelial shear stress. (c) Shear stress mapping for simulation including the deformational forces at 0.35 seconds or the height of the expansion. (d) Shear stress mapping for simulation including the deformational forces at 0.7.

the deformational forces. In conventional simulations that ignore the time-varying curvature change, such changes in endothelial shear stress are not accounted for.

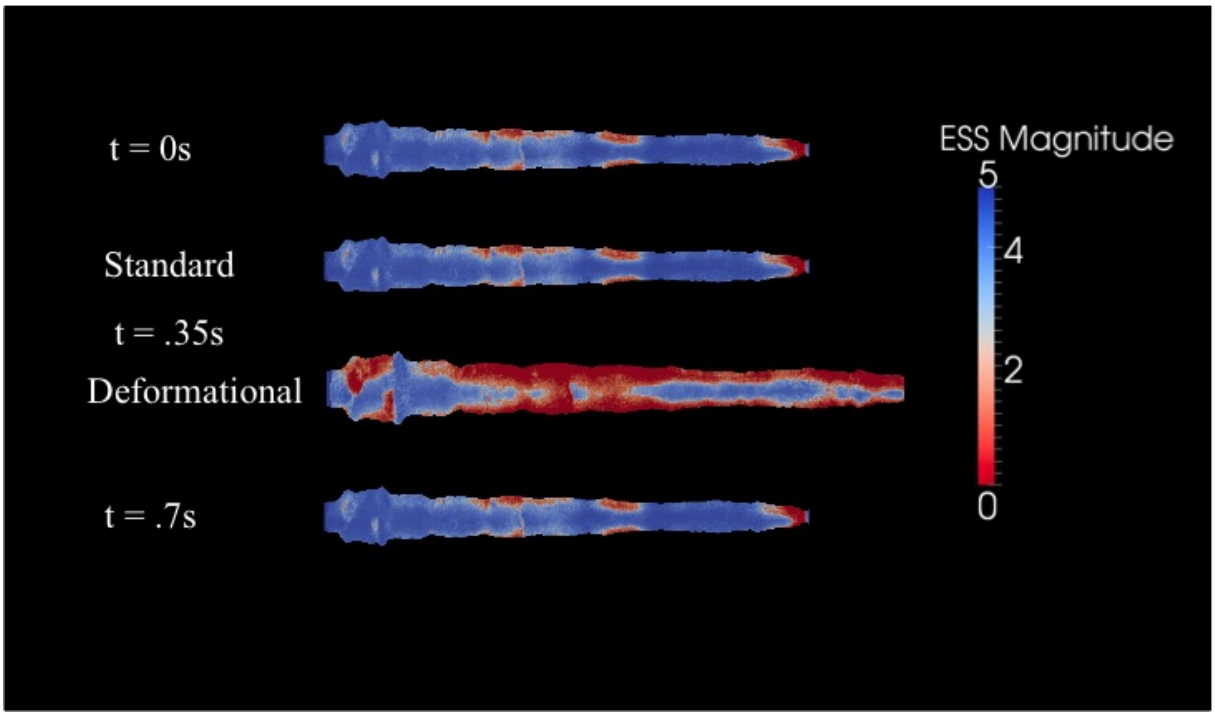


Figure 8.3: 2D projection of the shear stress map of the LAD artery at different points in the cardiac cycle. For the time point at $0.35s$, both the result from the standard force and the result when including the deformational forces are shown.

8.5 Discussion

In the current chapter, a method for accounting for the deformational forces that impact coronary blood flow due to the curvature change of the arteries is introduced. This method captures the dynamic curvature change while allowing the geometry of the artery itself to remain static. Key advantages are the ease of incorporating this force into the LBM as an external body force and removing the need to redefine the geometrical mesh at each time point. The use of the body force allows the simulation to be based on one 3D patient geometry, thus minimizing the number and duration of image acquisitions (and consequently minimizing the associated radiation dose)

and also reducing the computational complexity of the method. First, the flow in a simple geometry representing a coronary artery shows the successful addition of a sinusoidal curvature change. The simplified arterial motion allowed the focus to be on the unsteady motion on the arteries as the underlying myocardium expands and contracts. It was shown that the inclusion of this force caused a large temporal variance in the velocity profile. Second, the change in endothelial shear stress in a complex patient geometry was assessed. Again the curvature change was introduced through a sinusoidal wave and a strong qualitative change in ESS was observed. When comparing the result of this method to simulations including only a standard body force for steady flow, a large impact is shown.

In general, such computational studies offer a method of assessing the impact of various forces on ESS in real patient geometries. In this study, the focus was only on the addition of the deformational forces. In future work, the combination of pulsatile flow and the motion of the arteries should be considered. This can be especially important if cardiac disease is present. In such cases, the cardiac muscle may contract differently leading to potentially non-uniform curvature changes. Moreover, if aortic regurgitation is present, the phase difference between the arterial motion and pulsatile flow may increase having a severe effect on the overall shear rates ([94], [120]).

A limitation of this study is the assumption that the cardiac muscle is subjected to a uniform contraction and expansion across its volume should be modified to reflect more realistic change. The variation of curvature change across porcine hearts was

investigated in [142]. Similar changes for human myocardium should be studied and potentially included in the model. An ellipsoidal representation could be used to get a more accurate model. Also, patient-specific or more accurate models for the flow rate waveform could be used instead of the sinusoidal wave used here.

9

Proposed Future Work

> *Attention in computational vascular mechanics should focus on patient-specific analyses of disease progression, device-tissue interactions, and interventional and surgical planning based on appropriate couplings of advection-reactiondiffusion formulations and fluidsolid-growth models. Models should be as simple as possible, yet include complexities that enable the underlying mechanobiology and chemomechanics to be modeled well.*
>
> – Charles Taylor and Jay Humphrey [151]

The objective of the research presented in this thesis is to provide conceptual insight into the development of CVD and to serve as a proof of principle for larger circulatory models to aid in disease prediction and diagnosis. The prediction of disease localization and progression is still an important open question facing cardiovascular researchers. One of the fundamental questions is to understand the mechanism of cell movement through the vascular system and the likelihood of penetration of the vessel wall. Depending on the cell-types involved, these adhesions can lead to the development of an atherosclerotic plaque. During the past few decades, clinical studies have

found that *in silico* identification of areas of low ESS has enabled the potential for early detection of regions prone to atherosclerotic disease development. It is therefore of great importance to detect the underlying mechanisms at early stages in the disease process, so that appropriate treatment can be started in time.

Simulation of the movement of disease-specific cells through the blood vessels will enable the investigation of factors like cell concentration, cell size, blood pressure, and velocity and the calculation of the probability of adhesion at different points in the circulatory system. Such studies will provide insight into likely locations of disease sites and a coherent understanding of the complex interactions involved with the development of vulnerable plaques. However, the ability to model fluid movement in the full circulatory system is currently limited by deep conceptual and technical challenges. While computational models allow us to reconstruct patient specific arterial geometries and assess fluid flow in the system, these simulations typically involve a small subsection of the circulatory system or ignore complex cellular interactions.

In Chapter 4, the first multiscale simulation of cardiovascular flows in realistic human arterial geometries was presented. A challenge raised by this work was that even taking the described strategies for efficient parallelization, the simulation of one heartbeat required the use of 163,840 cores for a full six hours. This order of time scale is simply not feasible to enable physicians to leverage such personalized computer simulations as routine component of patient assessment. In the following chapters, methods to extend the accuracy of the model, improve computational efficiency, and

reduce time to solution through coupled space-time parallelism were presented. This work has raised several open questions for future research.

One such question involves the red blood cell model. In response to the simulation time exhibited in Chapter 4, the rest of this thesis focused on improving the fluid component of the model. The result is HARVEY, a highly efficient lattice Boltzmann based code, which is capable of accurately modeling blood flow for long time durations. Now that the fluid component has been optimized and extended to regimes beyond the continuum limit, can a red blood cell model be introduced that allows for reasonable time to solution for large-scale models? Moreover, mechanical factors like wall shear stress play a factor in its development and high low-density lipoprotein (LDL) concentration has been shown to correspond with sites of plaque development and indicate the nature of the plaque's growth [148], [106]. Through modeling the interaction of the LDLs with other cells and the arterial wall, quantities like concentration and residence time at the wall can be compared with experimental data for insight to the disease progression. In the initial work discussed here, red blood cells were modeled as rigid body ellipsoids. To study LDL interaction at the arterial wall, the interaction of the cells with the wall and with others cells plays an important role rather than in the previous case when the focus was on the bulk movement of the fluid. A low-dimensional model that takes into account the flexibility of the cells is necessary; however, the scale of the cells needed for the simulations is on the order of billions of cells simultaneously being modeled.

Another open question is whether or not these methods can extended to enable modeling of larger circulatory models. What are the bounds on the size an duration of the hemodynamic simulations? Typical hemodynamic models are simulated for short stretches of a single artery or focus only on the bulk fluid in an arterial system of typically 10-12 vessels. Recently, simulations of full body hemodynamics have started to be completed. In these cases, only fluid is modeled and only larger arteries are included (c.f [168], [105]).

In Chapter 8, a method to account for the deformational force applied to coronary blood flow as a result of the artery's curvature change. This was completed through the formalism of an external body force which both reduced the potential computational complexity of the model and the radiation dose the patient was exposed to by enabling arterial motion to be included even while the geometry remained static over the entire cardiac cycle. In all of the simulations discusses in that chapter, steady flow was used to emphasize the role curvature change had in changing the ESS exhibited in the arteries. In future work, a more realistic waveform to describe the curvature change as well as the coupling between the vessel movement and pulsatile flow should be investigated. Studies taking into account the potential phase difference between the arterial motion and flow rate should be pursued.

10

Conclusions

Each problem that I solved became a rule which served afterwards to solve other problems.

– René Descartes [34]

And so I conclude that blood lives and is nourished of itself and in no way depends on any other part of the body as being prior to it or more excellent... So that from this we may perceive the causes not only of life in general... but also of longer or shorter life, of sleeping and waking, of skill, of strength and so forth.

– William Harvey, *De motu cordis [68]*

This thesis has attempted to address the challenges associated with using massively parallel supercomputers to model blood flow in real patient geometries. The entire heart circulation system was simulated at high resolution using up to 294,912 processors of the IBM Blue Gene/P supercomputer. This involved the designing of new algorithms to address difficulties like the extremely complex and irregular geometry and workload balancing across the large core count.

Methods to optimize the the lattice Boltzmann model for the D3Q19 velocity

model as well as higher order models that extend the accuracy of the simulation beyond the continuum limit were presented. Techniques such as deep halo level ghost cells and hybrid programming models were introduced to help reduce the computational impact of these extended models.

In order to further reduce the overall time to solution for cardiovascular simulations, a novel scheme for coupling temporal and spatial decomposition was developed. This has the benefit of overcoming the fundamental strong scaling limit of space-parallel CFD methods and enabling more efficient use of the core counts becoming available on next generation systems.

Additionally, a new algorithm was presented to account for the arterial curvature change over the coarse of the cardiac cycle and its impact on coronary blood flow. Various steady and unsteady numerical simulations were performed, all yielding excellent agreement with either analytical solutions or *in vivo measurements*. The ability to evaluate the proclivity of patients for various cardiovascular diseases through personalized computer simulations has been demonstrated. In conclusion, the lattice Boltzmann methods discussed here are shown to be accurate and robust solvers for computational fluid dynamics in the regimes associated with cardiovascular disease. The work in this thesis demonstrates that applying such computational techniques to study CVD has the potential to ultimately help reduce long-term morbidity by identifying risk factors before there are clinical manifestations.

Bibliography

[1] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485. ACM, 1967.

[2] T. Aoki, H.A. Wood, L.J. Old, and E.A. Boyse. Arterial wall shear and distribution of early atheroma in man. *Nature*, 223(1):159, 1969.

[3] E.B. Arkilic, M.A. Schmidt, and K.S. Breuer. Gaseous slip flow in long microchannels. *Microelectromechanical Systems, Journal of*, 6(2):167 –178, jun 1997.

[4] E.J. Arlemark, S.K. Dadzie, and J.M. Reese. An extension to the Navier–Stokes equations to incorporate gas molecular collisions with boundaries. *Journal of Heat Transfer*, 132(4):041006, 2010.

[5] AMM Artoli. *Mesoscopic computational haemodynamics*. Ponsen en Looijen, 2003.

[6] T. Asakura and T. Karino. Flow patterns and spatial distribution of atherosclerotic lesions in human coronary arteries. *Circulation research*, 66(4):1045–1066, 1990.

[7] L. Axner, J. Bernsdorf, T. Zeiser, P. Lammers, J. Linxweiler, and A.G. Hoekstra. Performance evaluation of a parallel sparse lattice Boltzmann solver. *Journal of Computational Physics*, 227(10):4895–4911, 2008.

[8] J.A. Bærentzen and H. Anaæs. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):243–253, May 2005.

[9] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Parallel-in-time molecular-dynamics simulations. *Phys. Rev. E*, 66:057701, Nov 2002.

[10] G. Bal and Y. Maday. A parareal time discretization for non-linear p.d.e's with application to the pricing of an american put. *Lecture Notes in Computational Science and Engineering*, 23:189–202, 2002.

[11] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53(3):484–512, 1984.

[12] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras. A flexible high-performance lattice Boltzmann gpu code for the simulations of fluid flows

in complex geometries. *Concurrency and Computation: Practice and Experience*, 22(1):1–14, 2009.

[13] M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, E. Kaxiras, and J.K. Sircar. Muphy: A parallel MUltiPHYsics/scale code for high performance bio-fluidic simulations. *Computer Physics Communications*, 180(9):1495–1502, 2009.

[14] Jeremy Bernstein. *Kinetic theory in the expanding universe*. Cambridge University Press, 2004.

[15] L.A. Berry, W. Elwasif, J.M. Reynolds-Barredo, D. Samaddar, R. Sanchez, and D.E. Newman. Event-based parareal: A data-flow based implementation of parareal. *Journal of Computational Physics*, 231(17):5945 – 5954, 2012.

[16] BK Bharadvaj, RF Mabon, and DP Giddens. Steady flow in a model of the human carotid bifurcation. part iilaser-doppler anemometer measurements. *Journal of Biomechanics*, 15(5):363–378, 1982.

[17] P.L. Bhatnagar, E. Gross, and M. Krook. A model for collision processes in gases. *Physics Review Letters*, 94:511, 1954.

[18] M. Borkin, Krzysztof Gajos, A. Peters, D. Mitsouras, S. Melchionna, Frank Rybicki, C.L. Feldman, and H. Pfister. Evaluation of artery visualizations for heart disease diagnosis. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2479–2488, 2011.

[19] J Boyd and JM Buick. Three-dimensional modeling of the human carotid artery using the lattice Boltzmann method: Ii. shear analysis. *Physics in Medicine and Biology*, 53(20):5781, 2008.

[20] David L Brown, John Bell, Donald Estep, William Gropp, Bruce Hendrickson, Sallie Keller-McNulty, David Keyes, J Tinsley Oden, Linda Petzold, and Margaret Wright. Applied mathematics at the us department of energy: Past, present and a view to the future. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2008.

[21] M. Cannataro, Pietro H. Guzzi, G. Tradigo, and P. Veltri. A tool for the semiautomatic acquisition of the morphological data of blood vessel networks. In *Parallel and Distributed Processing with Applications, 2008. ISPA'08. International Symposium on*, pages 837–840. IEEE, 2008.

[22] Colin Gerald Caro, TJ Pedley, RC Schroter, and WA Seed. *The mechanics of the circulation*. Cambridge University Press, 2011.

[23] J. Carter, M. Soe, L. Oliker, Y. Tsuda, G. Vahala, L. Vahala, and A. Macnab. Magnetohydrodynamic turbulence simulations on the earth simulator using the lattice Boltzmann method. In *Proceedings of the 2005 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '05. IEEE Computer Society, 2005.

[24] Y.S. Chatzizisis, M. Jonas, A.U. Coskun, R. Beigel, B.V. Stone, C. Maynard,

R.G. Gerrity, W. Daley, C. Rogers, E.R. Edelman, et al. Prediction of the localization of high-risk coronary atherosclerotic plaques on the basis of low endothelial shear stress an intravascular ultrasound and histopathology natural history study. *Circulation*, 117(8):993–1002, 2008.

[25] D. Chen, N.A Eisley, P. Heidelberger, R.M. Senger, Y. Sugaware, S. Kumar, V. Salapura, D.L. Satterfield, B. Burrow-Steinmacher, and J. Parker. The IBM Blue Gene/q interconnection network and message unit. In *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11. IEEE Computer Society, 2011.

[26] H. Chen, S.A. Orszag, and I. Staroselsky. Macroscopic description of arbitrary Knudsen number flow using Boltzmann-BGK kinetic theory. *Journal of Fluid Mechanics*, 574:495–505, 2007.

[27] Shiyi Chen and Gary D Doolen. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, 30(1):329–364, 1998.

[28] C. Cheng, D. Tempel, R. van Haperen, A. van der Baan, F. Grosveld, M. Daemen, R. Krams, and R. de Crom. Atherosclerotic lesion size and vulnerability are determined by patterns of fluid shear stress. *Circulation*, 113(23):2744–2753, 2006.

[29] C. Chevalier and F. Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6):318–331, 2008.

[30] J.R. Clausen, D.A. Reasor Jr, and C.K. Aidun. Parallel performance of a lattice-Boltzmann/finite element cellular blood flow solver on the ibm Blue Gene/p architecture. *Computer Physics Communications*, 181(6):1013–1020, 2010.

[31] S. Colin. Rarefaction and compressibility effects on steady and transient gas flows in microchannels. *Microfluidics and Nanofluidics*, 1:268–279, 2005. 10.1007/s10404-004-0002-y.

[32] Michael E DeBakey, Gerald M Lawrie, and Donald H Glaeser. Patterns of atherosclerosis and their surgical significance. *Annals of surgery*, 201(2):115, 1985.

[33] L.A. DeRose. The hardware performance monitor toolkit. In *Euro-Par 2001 Parallel Processing*, pages 122–132. Springer, 2001.

[34] René Descartes and John Veitch. *Discourse on method*. Open Court, 1962.

[35] A. Dullweber, B. Leimkuhler, and R. McLachlan. Symplectic splitting methods for rigid body molecular dynamics. *The Journal of Chemical Physics*, 107:5840, 1997.

[36] M.M. Dupin, I. Halliday, C.M. Care, L. Alboul, and L.L. Munn. Modeling the flow of dense suspensions of deformable particles in three dimensions. *Physical Review E*, 75(6):066707, 2007.

[37] A. Dupuis and B. Chopard. Theory and applications of an alternative lattice Boltzmann grid refinement algorithm. *Physical Review E*, 67(6):066707, 2003.

[38] CD Eggleton and AS Popel. Large deformation of red blood cell ghosts in a simple shear flow. *Physics of Fluids*, 10:1834, 1998.

[39] J Ehlers and W Rindler. Robertson-walker metric? *Astron. Astrophys*, 174:14, 1987.

[40] Arthur Ellis. *Teaching and Learning Elementary Social Studies*. 1970.

[41] Franklin H Epstein and H Franklin Bunn. Pathogenesis and treatment of sickle cell disease. *New England Journal of Medicine*, 337(11):762–769, 1997.

[42] D.J.W. Evans, P.V. Lawford, J. Gunn, D. Walker, D.R. Hose, RH Smallwood, B Chopard, M Krafczyk, J Bernsdorf, and A Hoekstra. The application of multiscale modeling to the process of development and prevention of stenosis in a stented coronary artery. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1879):3343–3360, 2008.

[43] Q. Fan and H. Xue. Compressible effects in microchannel flows [MEMS]. In *Electronics Packaging Technology Conference, 1998. Proceedings of 2nd*, pages 224 –228, dec 1998.

[44] C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications.

International Journal for Numerical Methods in Engineering, 58(9):1397–1434, 2003.

[45] C. Farhat, J. Cortial, C. Dastillung, and H. Bavestrello. Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses. *International Journal for Numerical Methods in Engineering*, 67(5):697–724, 2006.

[46] Alberto Figueroa, Tommaso Mansi, Puneet Sharma, and Nathan Wilson. CFD challenge: Simulation of hemodynamics in a patient-specific aortic coarctation model. `http://www.vascularmodel.org/miccai2012/`, 2012.

[47] O. Filippova and D. Hänel. Grid refinement for lattice-BGK models. *Journal of Computational Physics*, 147(1):219–228, 1998.

[48] O. Filippova and D. Hänel. Acceleration of lattice-BGK schemes with grid refinement. *Journal of Computational Physics*, 165(2):407–427, 2000.

[49] O. Filippova, S. Succi, F. Mazzocco, C. Arrighetti, G. Bella, and D. Hänel. Multiscale lattice Boltzmann schemes with turbulence modeling. *Journal of Computational Physics*, 170(2):812–829, 2001.

[50] P.F. Fischer, F. Hecht, and Y. Maday. A parareal in time semi-implicit approximation of the Navier-Stokes equations. In *Proceedings of Fifteen International*

Conference on Domain Decomposition Methods, pages 433–440. Springer Verlag, 2004.

[51] M.H. Friedman, B.D. Kuban, P. Schmalbrock, K. Smith, and T. Altan. Fabrication of vascular replicas from magnetic resonance images. *Journal of Biomechanical Engineering*, 117(3):364, 1995.

[52] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier-stokes equation. *Physical Review Letters*, 56(14):1505–1508, 1986.

[53] A. Gara, M. A. Blumrich, and et. al. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49:193–212, 2005.

[54] A. Gara, M.A. Blumrich, D. Chen, G.L.T. Chiu, P. Coteus, M.E. Giampapa, R.A. Haring, P. Heidelberger, D. Hoenicke, G.V. Kopcsay, et al. Overview of the Blue Gene/l system architecture. *IBM Journal of Research and Development*, 49(2.3):195–212, 2005.

[55] J.G. Gay and B.J. Berne. Modification of the overlap potential to mimic a linear site–site potential. *The Journal of Chemical Physics*, 74(6):3316–3319, 1981.

[56] I.P. Gent, C. Jefferson, and I. Miguel. Minion: A fast scalable constraint solver. *Frontiers in Artificial Intelligence and Applications*, 141:98, 2006.

[57] DG Gibson, TA Prewitt, and DJ Brown. Analysis of left ventricular wall movement during isovolumic relaxation and its relation to coronary artery disease. *British Heart Journal*, 38(10):1010–1019, 1976.

[58] M. Gilge. Ibm system Blue Gene solution: Blue Gene/Q application development. *IBM Redbook Draft SG24–7948–00*, 2012.

[59] S. Glagov, C. Zarins, D.P. Giddens, D.N. Ku, et al. Hemodynamics and atherosclerosis. insights and perspectives gained from studies of human arteries. *Archives of Pathology & Laboratory Medicine*, 112(10):1018, 1988.

[60] J. Götz. *Numerical simulation of blood flow with lattice Boltzmann methods*. PhD thesis, Masters thesis, University of Erlangen-Nuremberg, Computer Science 10–System Simulation, 2006.

[61] J. Götz, K. Iglberger, C. Feichtinger, S. Donath, and U. Rüde. Coupling multibody dynamics and computational fluid dynamics on 8192 processor cores. *Parallel Computing*, 36(2):142–151, 2010.

[62] L. Grinberg, T. Anor, E. Cheever, J.R. Madsen, and G.Em. Karniadakis. Simulation of the human intracranial arterial tree. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1896):2371–2386, 2009.

[63] L. Grinberg, T. Anor, J.R. Madsen, A. Yakhot, and G.E. Karniadakis. Larg.e.-

scale simulation of the human arterial tree. *Clinical and Experimental Pharmacology and Physiology*, 36(2):194–205, 2009.

[64] L. Grinberg, V.i Morozov, D. Fedosov, J.A. Insley, M.E. Papka, K. Kumaran, and G.E. Karniadakis. A new computational paradigm in multiscale simulations: Application to brain blood flow. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12. IEEE, 2011.

[65] M.F. Gross and M.H. Friedman. Dynamics of coronary artery curvature obtained from biplane cineangiograms. *Journal of Biomechanics*, 31(5):479–484, 1998.

[66] John L Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[67] R.A. Haring, M. Ohmacht, T.W. Fox, M.K. Gschwind, D.L. Satterfield, K. Sugavanam, P.W. Coteus, P. Heidelberger, M.A. Blumrich, R.W. Wisniewski, A. Gara, G. Liang-Tai Chiu, P.A. Boyle, N.H. Chist, and C. Kim. The IBM Blue Gene/q compute chip. volume 32, pages 48–60, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[68] William Harvey. *De Motu Cordis*. Frankfurt: William Fitzer, 1628.

[69] M. Hecht and J. Harting. Implementation of on-site velocity boundary condi-

tions for d3q19 lattice Boltzmann simulations. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(01):P01018, 2010.

[70] A. Henderson. Paraview guide, a parallel visualization application. 2007.

[71] P. Herscovitch and M.E Raichle. What is the correct value for the brain-blood partition coefficient for water? *Journal of Cerebral Blood Flow & Metabolism*, 5(1):65–69, 1985.

[72] FJ Higuera, S. Succi, and R. Benzi. Lattice gas dynamics with enhanced collisions. *EPL (Europhysics Letters)*, 9(4):345–349, 1989.

[73] PJ Hoogerbrugge and JMVA Koelman. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *EPL (Europhysics Letters)*, 19(3):155, 1992.

[74] IBM Blue Gene Team. Overview of the IBM Blue Gene/p project. *IBM Journal of Research and Development*, 52:1–2, 2008.

[75] G. Karypis and V. Kumar. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[76] F. Kjolstad and M. Snir. Ghost Cell Pattern. In *2nd Annual Workshop on Parallel Programming Patterns, ParaPLoP'10*. Association for Computing Machinery (ACM), Mar. 2010.

[77] N. Kok Fu and N.H.J. Mohd Ali. Improving pipelined time stepping algorithm for distributed memory multicomputers. *Sains Malaysiana*, 39(6):1041–1048, 2010.

[78] B-K Koo et al. Diagnosis of ischemia-causing stenoses obtained via non-invasive fractional flow reserve (discover-flow): a prospective multicentre first-in-man study. In *Proceedings of EuroPCR: the annual meeting of the European Association for Percutaneous Cardiovascular Interventions.*, PCR '11, 2011.

[79] C. Körner, T. Pohl, U. Rüde, N. Thürey, and T. Zeiser. Parallel lattice Boltzmann methods for cfd applications. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 439–466. Springer, 2006.

[80] R Krams, JJ Wentzel, JAF Oomen, R Vinke, JCH Schuurbiers, PJ De Feyter, PW Serruys, and CJ Slager. Evaluation of endothelial shear stress and 3d geometry as factors determining the development of atherosclerosis and remodeling in human coronary arteries in vivo combining 3d reconstruction from angiography and i.v.us (angus) with computational fluid dynamics. *Arteriosclerosis, thrombosis, and vascular biology*, 17(10):2061–2065, 1997.

[81] J.F. Ladisa, C.A. Figueroa, I.E. Vignon-Clementel, H. Jin Kim, N. Xiao, L.M. Ellwein, F.P. Chan, J.A. Feinstein, C.A. Taylor, et al. Computational simulations for aortic coarctation: representative results from a sampling of patients. *Journal of biomechanical engineering*, 133(9):091008, 2011.

[82] G. Lakner, I.H. Chung, G. Cong, S. Fadden, N. Goracke, D. Klepacki, J. Lien, C. Pospiech, S.R. Seelam, and H.F. Wen. IBM system Blue Gene solution: Performance analysis tools. *IBM Redpaper Publication*, 2008.

[83] Andrea S Les, Shawn C Shadden, C Alberto Figueroa, Jinha M Park, Maureen M Tedesco, Robert J Herfkens, Ronald L Dalman, and Charles A Taylor. Quantification of hemodynamics in abdominal aortic aneurysms during rest and exercise using magnetic resonance imaging and computational fluid dynamics. *Annals of Biomedical Engineering*, 38(4):1288–1313, 2010.

[84] Aristid Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.

[85] J.-L. Lions, Y. Maday, and G. Turinici. A parareal in time discretization of P.D.E's. *C.R. Acad. Sci. Paris, Serie I*, 332:661–668, 2001.

[86] Y. Liu, L. Zhang, X. Wang, and W.K. Liu. Coupling of navier–stokes equations with protein molecular dynamics and its application to hemodynamics. *International Journal for Numerical Methods in Fluids*, 46(12):1237–1252, 2004.

[87] J. Z Ma, J. Ebben, H. Xia, and A.J. Collins. Hematocrit level and associated mortality in hemodialysis patients. *Journal of the American Society of Nephrology*, 10(3):610–619, 1999.

[88] R.M. Macmeccan, JR Clausen, GP Neitzel, and CK Aidun. Simulating deformable particle suspensions using a coupled lattice-boltzmann and finite-element method. *Journal of Fluid Mechanics*, 618(1):13–39, 2009.

[89] O. Malaspinas, B. Chopard, and J. Latt. General regularized boundary condition for multi-speed lattice Boltzmann models. *Computers & Fluids*, 49(1):29–35, 2011.

[90] A.M. Malek, S.L. Alper, and S. Izumo. Hemodynamic shear stress and its role in atherosclerosis. *JAMA: the Journal of the American Medical Association*, 282(21):2035–2042, 1999.

[91] FF Mark, CB Bargeron, OJ Deters, and MH Friedman. Nonquasi-steady character of pulsatile flow in human coronary arteries. *Journal of Biomechanical Engineering*, 107(1):24, 1985.

[92] N.S. Martys and H. Chen. Simulation of multicomponent fluids in complex three-dimensional geometries by the lattice Boltzmann method. *Physical Review E*, 53(1):743, 1996.

[93] N.S. Martys, X. Shan, and H. Chen. Evaluation of the external force term in the discrete Boltzmann equation. *Physical Review E*, 58(5):6855–6857, 1998.

[94] S. Matsuo, M. Tsuruta, M. Hayano, Y. Imamura, Y. Eguchi, T. Tokushima, and S. Tsuji. Phasic coronary artery flow velocity determined by doppler flowmeter

catheter in aortic stenosis and aortic regurgitation. *The American Jjournal of Cardiology*, 62(13):917–922, 1988.

[95] K. Mattila, J. Hyväluoma, T. Rossi, M. Aspnäs, and J. Westerholm. An efficient swap algorithm for the lattice Boltzmann method. *Computer Physics Communications*, 176(3):200–210, 2007.

[96] P.V. Mazzeo, M.D.and Coveney. Hemelb: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries. *Computer Physics Communications*, 178(12):894–914, 2008.

[97] G.R. McNamara and G. Zanetti. Use of the Boltzmann equation to simulate lattice-gas automata. *Physical Review Letters*, 61(20):2332–2335, 1988.

[98] S. Melchionna. A model for red blood cells in simulations of large-scale blood flows. *Macromolecular Theory and Simulations*, 20(7):548–561, 2011.

[99] S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F.J. Rybicki, D. Mitsouras, A.U. Coskun, and C.L. Feldman. Hydrokinetic approach to large-scale cardiovascular blood flow. *Computer Physics Communications*, 181(3):462–472, 2010.

[100] S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, Frank J. Rybicki, D. Mitsouras, Ahmet U. Coskun, and C.L. Feldman. Hydrokinetic approach to large-

scale cardiovascular blood flow. *Computer Physics Communications*, 181(3):462 – 472, 2010.

[101] S. Melchionna, J. Lätt, E. Kaxiras, A. Peters, M. Bernaschi, and S. Succi. Endothelial shear stress from large-scale blood flow simulations. In *Proceedings of Fifth European Conference on Computational Fluid Dynamics*, ECCOMAS CFD'10, 2010.

[102] M. Minion. A hybrid parareal spectral deferred corrections method. *Communications in Applied Mathematics and Computational Science*, 5:265–301, 2010.

[103] J.E. Moore, E.S. Weydahl, A. Santamarina, et al. Frequency dependence of dynamic curvature effects on flow through coronary arteries. *Journal of Biomechanical Engineering*, 123(2):129, 2001.

[104] et. al. N. R. Adiga, M. A. Blumrich. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49:265–276, 2005.

[105] M.L. Neal and R. Kerckhoffs. Current progress in patient-specific modeling. *Briefings in Bioinformatics*, 11(1):111–126, 2010.

[106] U. Olgac, D. Poulikakos, S.C. Saur, H. Alkadhi, and V. Kurtcuoglu. Patient-specific three-dimensional simulation of ldl accumulation in a human left coronary artery in its healthy and atherosclerotic states. *American Journal of Physiology-Heart and Circulatory Physiology*, 296(6):H1969–H1982, 2009.

[107] Catherine M Ong, Charles E Canter, Fernando R Gutierrez, Daniel R Sekarski, and David R Goldring. Increased stiffness and persistent narrowing of the aorta after successful repair of coarctation of the aorta: relationship to left ventricular mass and blood pressure at rest and with exercise. *American Heart Journal*, 123(6):1594–1600, 1992.

[108] R. Ouared and B. Chopard. Lattice Boltzmann simulations of blood flow: non-newtonian rheology and clotting processes. *Journal of statistical physics*, 121(1-2):209–221, 2005.

[109] W. Pan, D.A. Fedosov, B. Caswell, and G.E. Karniadakis. Predicting dynamics and rheology of blood flow: A comparative study of multiscale and low-dimensional models of red blood cells. *Microvascular Research*, 82(2):163–170, 2011.

[110] T.J. Pedley. *The fluid mechanics of large blood vessels*, volume 1. Cambridge University Press Cambridge, 1980.

[111] Max F Perutz. *I Wish Id Made You Angry Earlier: Essays on Science, Scientists, and Humanity*. Oxford University Press, 2002.

[112] C.S. Peskin. The immersed boundary method. *Acta numerica*, 11(0):479–517, 2002.

[113] A. Peters, S. Melchionna, E. Kaxiras, J. Lätt, J. Sircar, M. Bernaschi, M. Bis-

son, and S. Succi. Multiscale simulation of cardiovascular flows on the IBM Blue Gene/P: Full heart-circulation system at red-blood cell resolution. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10. IEEE Computer Society, 2010.

[114] A. Peters Randles, M. Baecher, H. Pfister, and E. Kaxiras. A lattice Boltzmann simulation of hemodynamics in a patient-specific aortic coarctation model. In *Proceedings of Statistical Atlases and Computational Models of the Heart (STACOM) Computational Fluid Dynamics Challenge*, 2012.

[115] A. Peters Randles, V. Kale, J.R. Hammond, W. Gropp, and E. Kaxiras. Performance analysis of the lattice Boltzmann model beyond Navier-Stokes. In *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium*, IP.D.PS '13, 2013.

[116] A. Peters Randles and E. Kaxiras. Parallel in time approximation of the lattice Boltzmann method for laminar flows. April 2013. (submitted).

[117] A. Peters Randles and E. Kaxiras. Parallel in time approximation of the lattice Boltzmann method for laminar flows. March 2013. (submitted).

[118] J.C. Phillips, G. Zheng, S. Kumar, and L.V. Kalé. Namd: Biomolecular simulation on thousands of processors. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 36–36. IEEE, 2002.

[119] I.V. Pivkin and G.E. Karniadakis. Accurate coarse-grained modeling of red blood cells. *Physical Review Letters*, 101(11):118105, 2008.

[120] I.V. Pivkin, P.D. Richardson, D.H. Laidlaw, and G.E. Karniadakis. Combined effects of pulsatile flow and dynamic curvature on wall shear stress in a coronary artery bifurcation model. *Journal of Biomechanics*, 38(6):1283–1290, 2005.

[121] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, and T. Zeiser. Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures. In *Proceedings of the 2004 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '04. IEEE Computer Society, 2004.

[122] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rde. Optimization and profiling of the cache performance of parallel lattice Boltzmann codes in 2D and 3D. *Parallel Processing Letters*, 13:2003, 2003.

[123] M. Prosi, K. Perktold, Z. Ding, M.H. Friedman, et al. Influence of curvature dynamics on pulsatile coronary artery flow in a realistic bifurcation model. *Journal of Biomechanics*, 37(11):1767, 2004.

[124] YH Qian, D d'Humieres, and P Lallemand. Lattice BGK models for navier-stokes equation. *EPL (Europhysics Letters)*, 17(6):479, 1992.

[125] Y. Qiu, J.M. Tarbell, et al. Numerical simulation of pulsatile flow in a compliant

curved tube model of a coronary artery. *Journal of Biomechanical Engineering*, 122(1):77, 2000.

[126] A. Quarteroni, A. Veneziani, and P. Zunino. Mathematical and numerical modeling of solute dynamics in blood flow and arterial walls. *SIAM Journal on Numerical Analysis*, 39(5):1488–1511, 2002.

[127] P. Rao. Coarctation of the aorta. *Current Cardiology Reports*, 7:425–434, 2005. 10.1007/s11886-005-0060-0.

[128] JM Reynolds-Barredo, DE Newman, JM Reynolds-Barredo, R. Sanchez, and LA Berry. modeling parareal convergence in 2D drift wave plasma turbulence. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 726–727. IEEE, 2012.

[129] JM Reynolds-Barredo, D.E. Newman, R. Sanchez, D. Samaddar, L.A. Berry, and W.R. Elwasif. Mechanisms for the convergence of time-parallelized, parareal turbulent plasma simulations. *Journal of Computational Physics*, 2012.

[130] Y. Richter and E.R. Edelman. Cardiology is flow. *Circulation*, 113(23):2679–2682, 2006.

[131] Robert Ringer. *Action!: Nothing Happens Until Something Moves*. M. Evans, 2004.

[132] H.P. Robertson. On the foundations of relativistic cosmology. *Proceedings of*

the National Academy of Sciences of the United States of America, 15(11):822, 1929.

[133] V.L Roger, A.S. Go, D.M. Lloyd-Jones, E.J. Benjamin, J.D. Berry, D.M. Borden, W.B .and Bravata, S. Dai, E.S. Ford, C.S. Fox, et al. Heart disease and stroke statistics2012 update a report from the american heart association. *Circulation*, 125(1):e2–e220, 2012.

[134] Eric Rosenthal. Stent implantation for aortic coarctation: the treatment of choice in adults?*. *Journal of the American College of Cardiology*, 38(5):1524–1527, 2001.

[135] HN Sabbah, FJ Walburn, and P.D. Stein. Patterns of flow in the left coronary artery. *Journal of Biomechanical Engineering*, 106(3):272, 1984.

[136] D. Samaddar, D.E. Newman, and R. Sánchez. Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. *Journal of Computational Physics*, 229(18):6558–6573, 2010.

[137] A. Santamarina, E. Weydahl, J.M. Siegel, and J.E. Moore. Computational analysis of flow in a curved tube model of the coronary arteries: effects of time-varying curvature. *Annals of Biomedical Engineering*, 26(6):944–954, 1998.

[138] X. Shan and X. He. Discretization of the velocity space in the solution of the Boltzmann equation. *Physical Review Letters*, 80(1):65–68, 1998.

[139] X. Shan, X. Yuan, and H. Chen. Kinetic theory representation of hydrodynamics: a way beyond the Navier-Stokes equation. *Journal of Fluid Mechanics*, 550:413–441, 2006.

[140] A. Sorger, M. Freitag, A. Shaporin, and J. Mehner. CFD analysis of viscous losses in complex microsystems. In *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pages 1 –4, march 2012.

[141] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, and P. Gibbon. A massively space-time parallel n-body solver. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 92:1–92:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[142] C. Stevens and P.J. Hunter. Sarcomere length changes in a 3d mathematical model of the pig ventricles. *Progress in Biophysics and Molecular Biology*, 82(1):229–241, 2003.

[143] J. Strony, A. Beaudoin, D. Brands, and B. Adelman. Analysis of shear stress and hemodynamic factors in a model of coronary artery stenosis and thrombosis. *American Journal of Physiology-Heart and Circulatory Physiology*, 265(5):H1787–H1796, 1993.

[144] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001.

[145] S. Succi, O. Filippova, G. Smith, and E. Kaxiras. Applying the lattice Boltzmann equation to multiscale fluid problems. *Computing in Science & Engineering*, 3(6):26–37, 2001.

[146] K. Suga, S. Takenaka, T. Kinjo, and S. Hyodo. LBM and MD simulations of a flow in a nano-porous medium. In *Proceedings of the 2nd Asian Symposium on Computational Heat Transfer and Fluid Flow*, ASCHT '09, pages 112–117, 2009.

[147] C. Sun and L.L Munn. Particulate nature of blood determines macroscopic rheology: a 2-d lattice Boltzmann analysis. *Biophysical Journal*, 88(3):1635–1645, 2005.

[148] N. Sun, N.B. Wood, and X.Y. Xu. Computational modeling of mass transport in large arteries. 2010.

[149] C. A. Taylor, M.T. Draney, J.P. Ku, D. Parker, B.N. Steele, K. Wang, and C.K. Zarins. Predictive medicine: computational techniques in therapeutic decision-making. *Computer Aided Surgery*, 4(5):231–247, 1999.

[150] C.A. Taylor, T. Hughes, and C.K. Zarins. Finite element modeling of blood flow in arteries. *Computer Methods in Applied Mechanics and Engineering*, 158(1):155–196, 1998.

[151] Charles A Taylor and JD Humphrey. Open problems in computational vascular

biomechanics: hemodynamics and arterial wall mechanics. *Computer Methods in Applied Mechanics and Engineering*, 198(45):3514–3523, 2009.

[152] R. Temam. *Navier-Stokes equations: theory and numerical analysis*, volume 343. Oxford University Press, 2001.

[153] M. Texon. A hemodynamic concept of atherosclerosis, with particular reference to coronary occlusion. *AMA Archives of Internal Medicine*, 99(3):418–427, 1957.

[154] Mano J Thubrikar and Francis Robicsek. Pressure-induced arterial wall stress and atherosclerosis. *The Annals of Thoracic Surgery*, 59(6):1594–1603, 1995.

[155] G.B. Thurston. Viscoelasticity of human blood. *Biophysical Journal*, 12(9):1205–1217, 1972.

[156] R. Torii, J. Keegan, N.B. Wood, A.W. Dowsey, A.D. Hughes, G.Z. Yang, D.N. Firmin, S.A. Thom, and X. Y. Xu. Mr image-based geometric and hemodynamic investigation of the right coronary artery with dynamic vessel motion. *Annals of Biomedical Engineering*, 38(8):2606–2620, 2010.

[157] R. Torii, J. Keegan, N.B. Wood, A.W. Dowsey, A.D. Hughes, G.Z. Yang, D.N. Firmin, S.A. Mcg Thom, and X.Y. Xu. The effect of dynamic vessel motion on haemodynamic parameters in the right coronary artery: a combined mr and cfd study. *British Journal of Radiology*, 82(Special Issue 1):S24–S32, 2009.

[158] Alan Mathison Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, B237:37–72, 1952.

[159] D.A. Vorp, D.A. Steinman, and C.R. Ethier. Computational modeling of arterial biomechanics. *Computing in Science & Engineering*, 3(5):51–64, 2001.

[160] D.A. Vorp, D.A. Steinman, and C.R. Ethier. Computational modeling of arterial biomechanics. *Computing in Science & Engineering*, 3(5):51–64, 2001.

[161] Patrick B Warren. Dissipative particle dynamics. *Current Opinion in Colloid & Interface Science*, 3(6):620–624, 1998.

[162] G. Wellein, T. Zeiser, G. Hager, and S. Donath. On the single processor performance of simple lattice Boltzmann kernels. *Computers & Fluids*, 35(8-9):910 – 919, 2006. Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science.

[163] N. Westerhof. *Snapshots of hemodynamics*. Springer, 2010.

[164] E.S. Weydahl and J.E. Moore. Dynamic curvature strongly affects wall shear rates in a coronary artery bifurcation model. *Journal of Biomechanics*, 34(9):1189–1196, 2001.

[165] S. Williams, L. Oliker, J. Carter, and J. Shalf. Extracting ultra-scale lattice Boltzmann performance via hierarchical and distributed auto-tuning. In *Proceedings of the 2011 ACM/IEEE International Conference for High Perfor-*

mance Computing, Networking, Storage and Analysis, SC '11, pages 1–10. IEEE Computer Society, 2011.

[166] M.M. Wintrobe, J.P. Greer, et al. *Wintrobe's Clinical Hematology*, volume 1. Lippincott Williams & Wilkins, 2009.

[167] N. Woolf. The origins of atherosclerosis. *Postgraduate Medical Journal*, 54(629):156–162, 1978.

[168] N. Xiao, J.D. Humphrey, and A.C. Figueroa. Multi-scale computational model of three-dimensional hemodynamics within a deformable full-body arterial network. *Journal of Computational Physics*, 2012.

[169] K. Xu and Z. Li. Microchannel flow in the slip regime: gas-kinetic BGK Burnett solutions. *Journal of Fluid Mechanics*, 513:87–110, 2004.

[170] PR Zarda, S Chien, and R Skalak. Elastic deformations of red blood cells. *Journal of Biomechanics*, 10(4):211–221, 1977.

[171] R. Zhang, X. Shan, and H. Chen. Efficient kinetic method for fluid simulation beyond the Navier-Stokes equation. *Physics Review E*, 74:1–7, 2006.

[172] D.P. Zipes and H.J. Wellens. Sudden cardiac death. *Circulation*, 98(21):2334–2351, 1998.

[173] Q. Zou and X. He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids*, 9:1591, 1997.